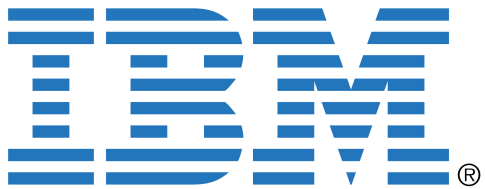
Rewriting Queries on SPARQL Views

Wangchao Le[1] Songyun Duan[2] Anastasios Kementsietsidis[2]
Feifei Li[1] Min Wang[3]

[1]Florida State University



[2]IBM T.J. Watson Research Center

[3]HP Lab China

March 24, 2011

Outline

Introduction

- Answering queries over *virtual* views is encountered in many settings.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

1. Views are materialized.

Introduction

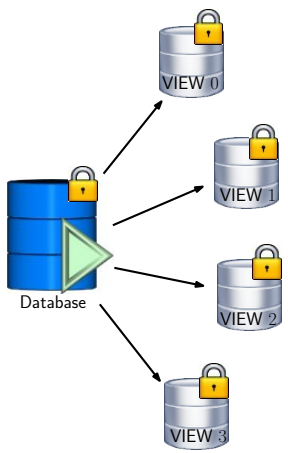
- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

1. Views are materialized.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

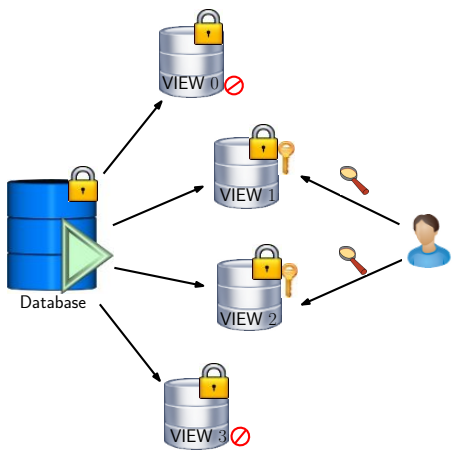
1. Views are materialized.



Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

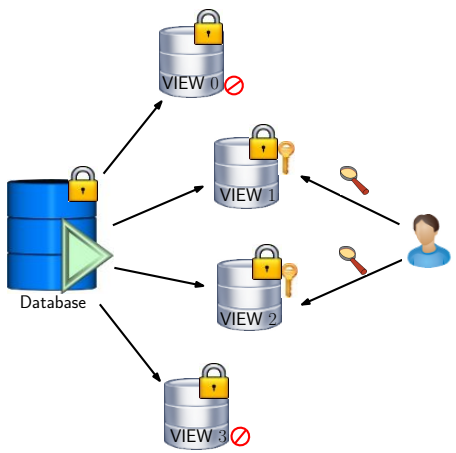
1. Views are materialized.



Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

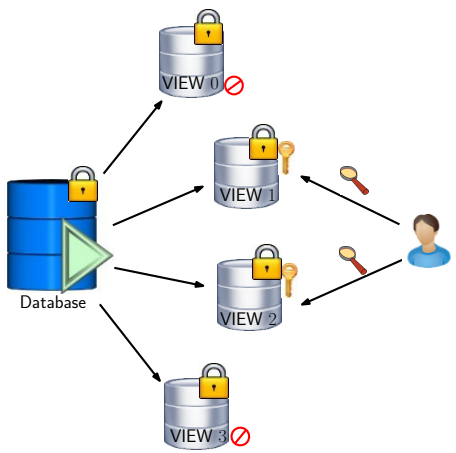
1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.



Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

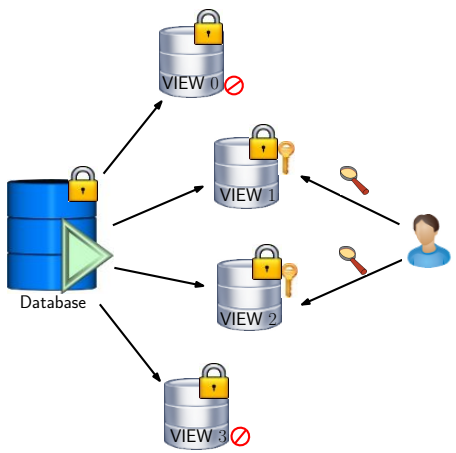
1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.



Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

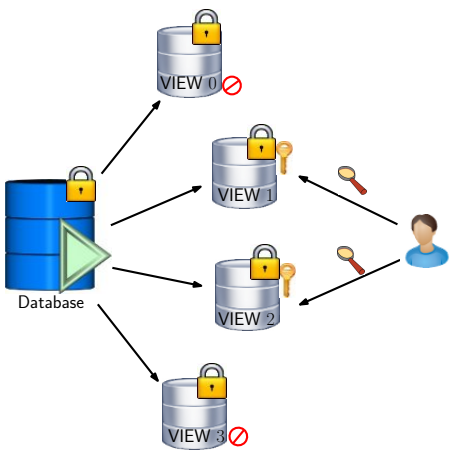
1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.
2. Views are virtual.



Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.

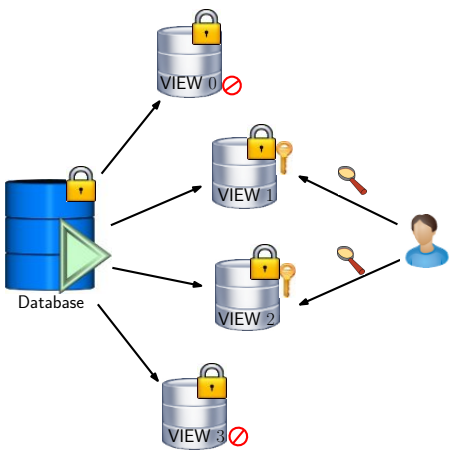


2. Views are virtual.

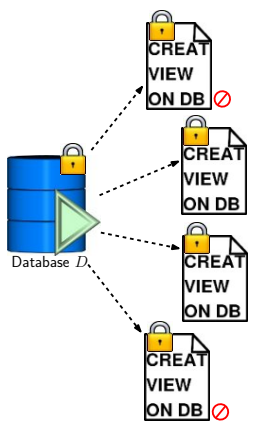
Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.



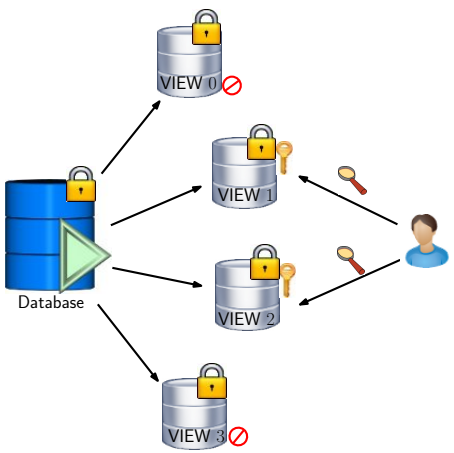
2. Views are virtual.



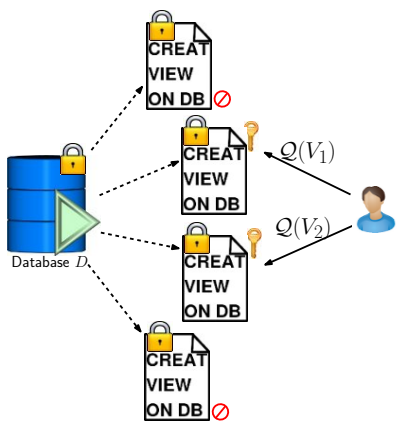
Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.



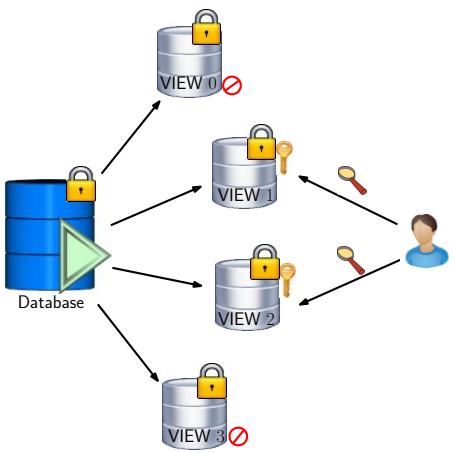
2. Views are virtual.



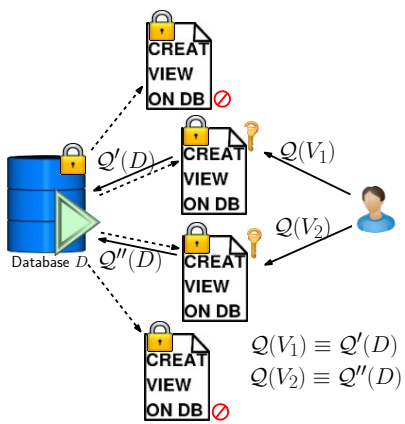
Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.



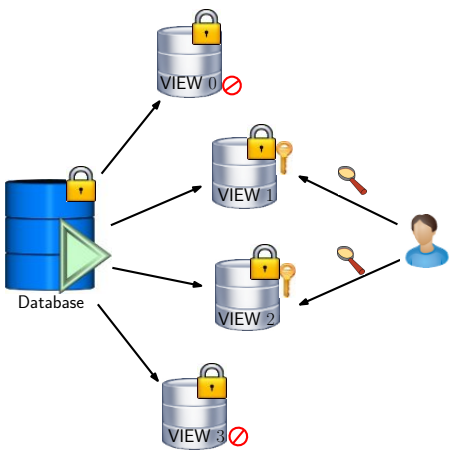
2. Views are virtual.



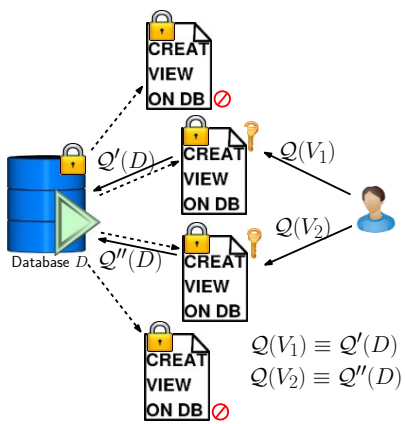
Introduction

- Answering queries over *virtual* views is encountered in many settings.
  - Access control.
  - Data integration.
- For example, DBAs use views to enforce security clearance.

1. Views are materialized.
   - $+$: Evaluation.
   - $-$: Space, updates, etc.



2. Views are virtual.
   - $+$: Maintenance, *e.g.*, ACID.
   - $-$: Rewriting is NP-hard.



Outline

Introduction: SPARQL query rewriting

- Answering query using views is well studied for relational DBs, but not well understood in RDF and SPARQL.
- Given a set definitions of views $\mathcal{V} = \{V_1, V_2, \ldots, V_l\}$ on a RDF graph G and SPARQL query Q over the views $\mathcal{V}$, compute a SPARQL query Q' such that $Q'(G) = Q(V(G))$.

Introduction: SPARQL query rewriting

- Answering query using views is well studied for relational DBs, but not well understood in RDF and SPARQL.
- Given a set definitions of views $\mathcal{V} = \{V_1, V_2, \ldots, V_l\}$ on a RDF graph G and SPARQL query Q over the views $\mathcal{V}$, compute a SPARQL query Q' such that $Q'(G) = Q(V(G))$.
- Soundness: The rewriting is sound iff $Q'(G) \subset Q(V(G))$.

Introduction: SPARQL query rewriting

- Answering query using views is well studied for relational DBs, but not well understood in RDF and SPARQL.
- Given a set definitions of views $\mathcal{V} = \{V_1, V_2, \ldots, V_l\}$ on a RDF graph G and SPARQL query Q over the views $\mathcal{V}$, compute a SPARQL query Q' such that $Q'(G) = Q(V(G))$.
- Soundness: The rewriting is sound iff $Q'(G) \subset Q(V(G))$.
- Completeness: The rewriting is complete iff $Q(V(G)) \subset Q'(G)$.

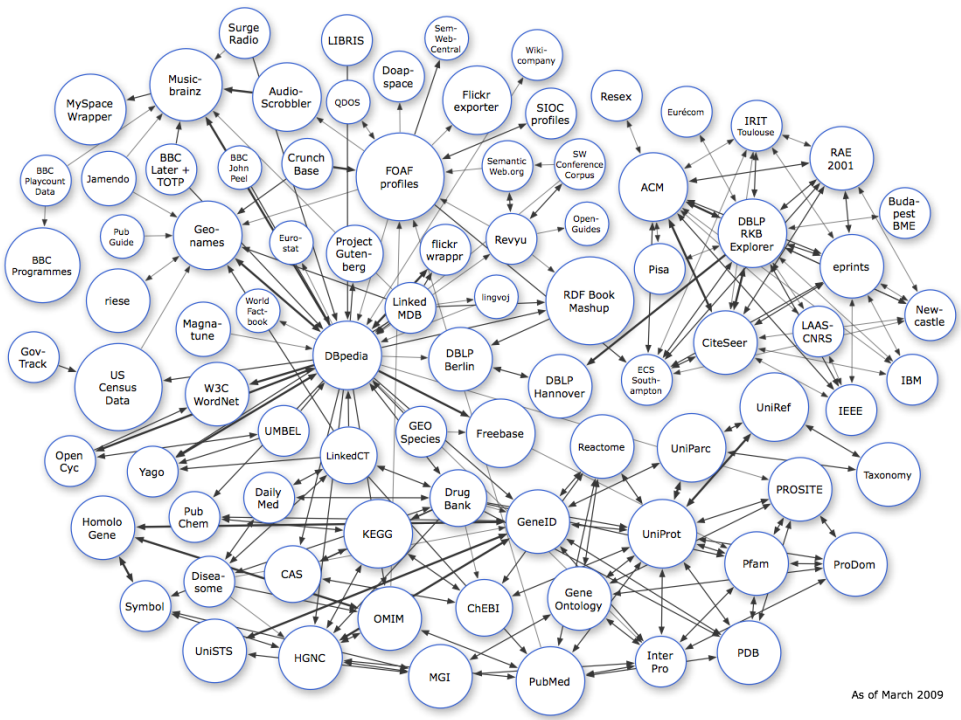
Outline

Introduction: RDF data and SPARQL language

- Resource Decription Framework (RDF) is commonly used in semantic web for knowledge representation.

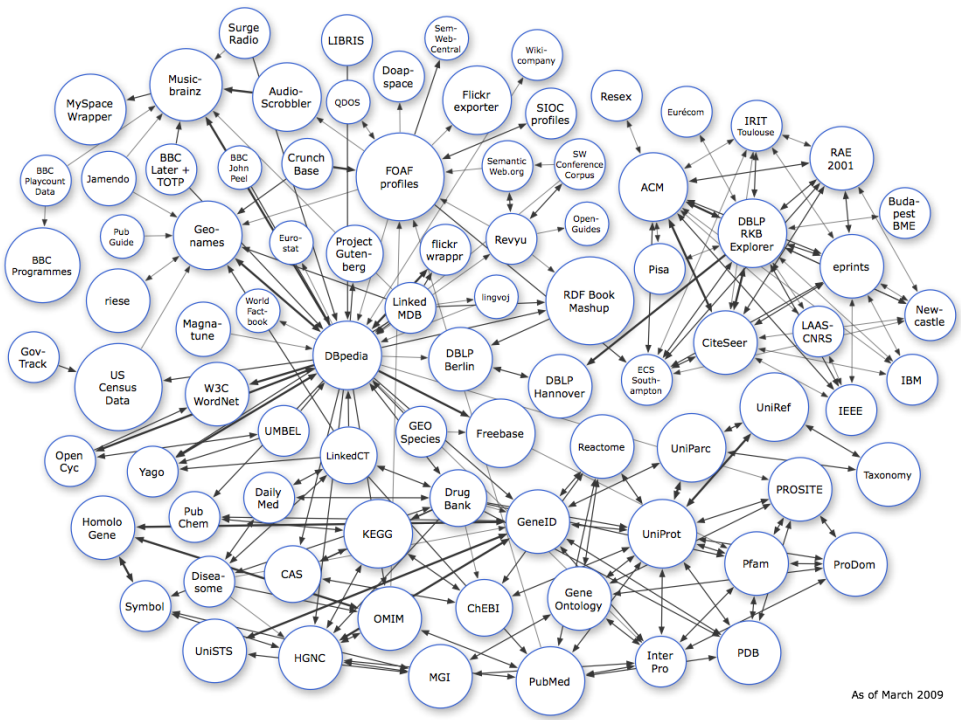
Introduction: RDF data and SPARQL language

- Resource Decription Framework (RDF) is commonly used in semantic web for knowledge representation.

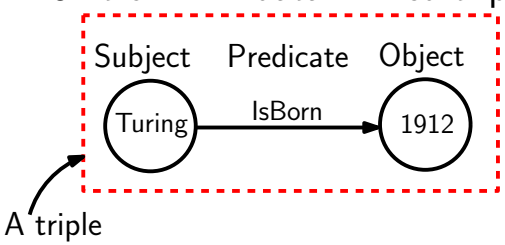


Introduction: RDF data and SPARQL language

- Resource Decription Framework (RDF) is commonly used in semantic web for knowledge representation.



- DBpedia extracted unstructured information and published in queryable structured format (RDF).
- Describing 3.5 million things.
- Unit of RDF data: linked triple.



Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
  $Q := (SELECT | CONSTRUCT) \quad RD \quad (WHERE \quad GP)$

Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
  $Q := (SELECT|CONSTRUCT) \quad RD \quad (WHERE \quad GP)$
- Given an RDF graph G and a graph pattern GP, Q searches for subgraphs in G that match GP.

Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
  $Q := (SELECT|CONSTRUCT) \quad RD \quad (WHERE \quad GP)$
- Given an RDF graph G and a graph pattern GP, Q searches for subgraphs in G that match GP.
- RD is the result description.

Introduction: RDF data and SPARQL language

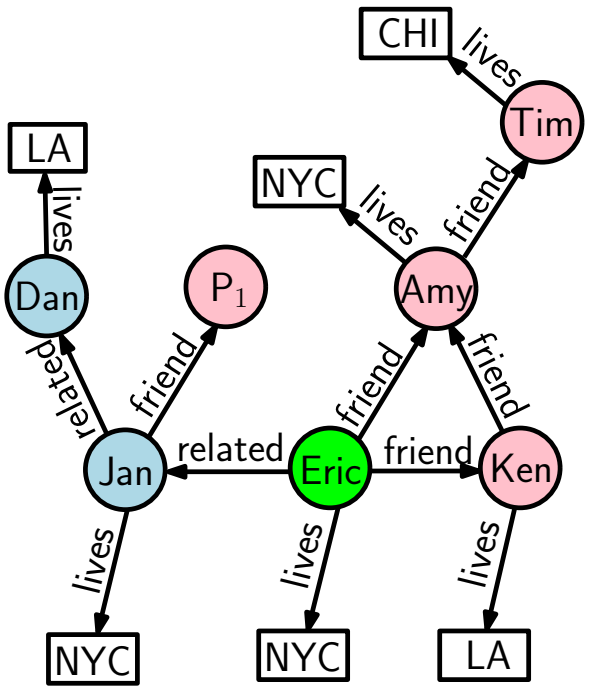
- We are interested in SPARQL queries of the form
  $Q := (SELECT|CONSTRUCT) \quad RD \quad (WHERE \quad GP)$
- Given an RDF graph G and a graph pattern GP, Q searches for subgraphs in G that match GP.
- RD is the result description.
  - For $SELECT$, it projects the values of variables (like SQL).
  - For $CONSTRUCT$, it is a set of triple templates that constructs a new RDF graph from the matched subgraphs in G.

Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
  $Q := (SELECT|CONSTRUCT) \quad RD \quad (WHERE \quad GP)$
- Given an RDF graph G and a graph pattern GP, Q searches for subgraphs in G that match GP.
- RD is the result description.
  - For $SELECT$, it projects the values of variables (like SQL).
  - For $CONSTRUCT$, it is a set of triple templates that constructs a new RDF graph from the matched subgraphs in G.
- Finally, there is a Boolean SPARQL query of form: $ASK \ GP$, which indicates if GP exists, or not, in G.

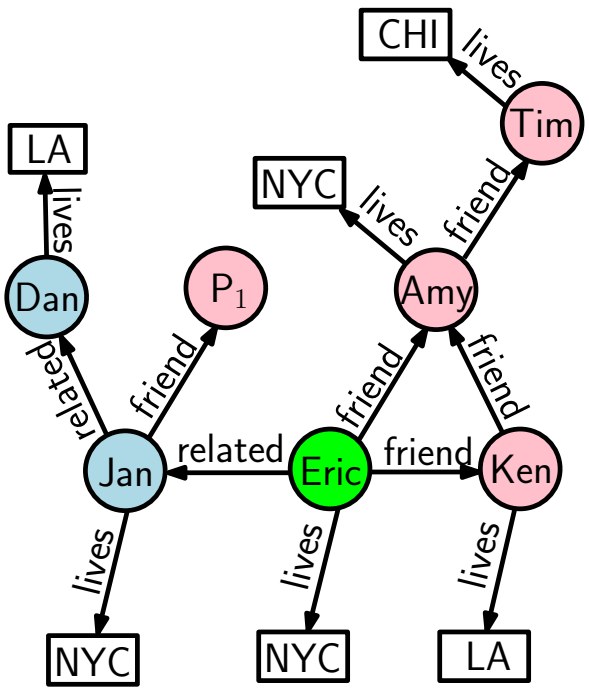
Outline

Introduction: A running example and solution outline



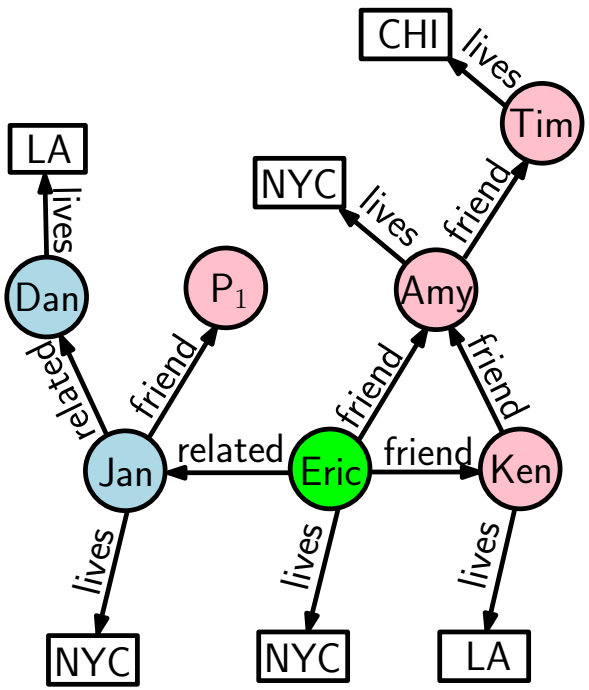
Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.



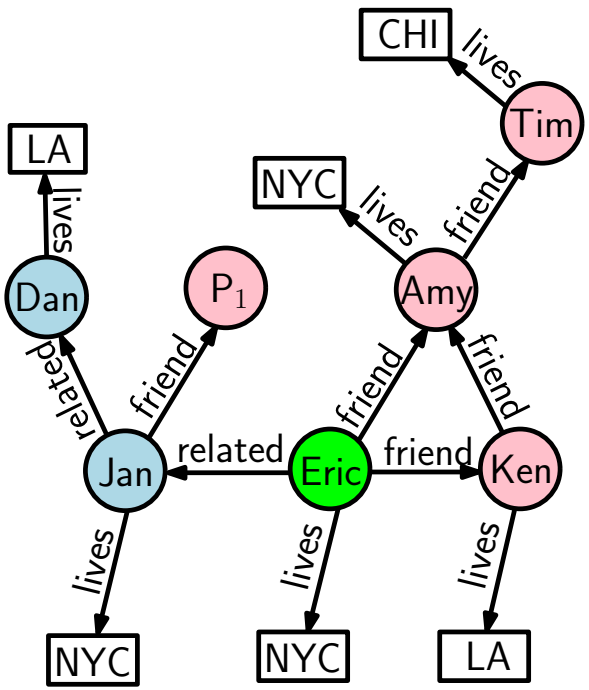
Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



Introduction: A running example and solution outline

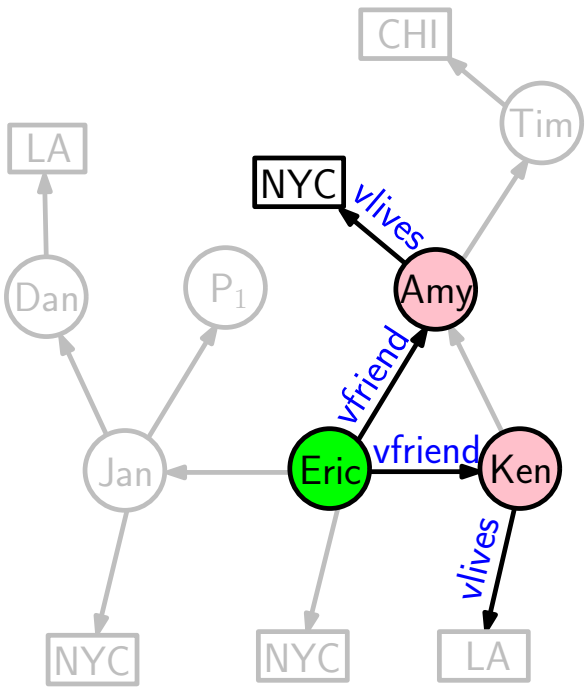
- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F:
CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

Introduction: A running example and solution outline

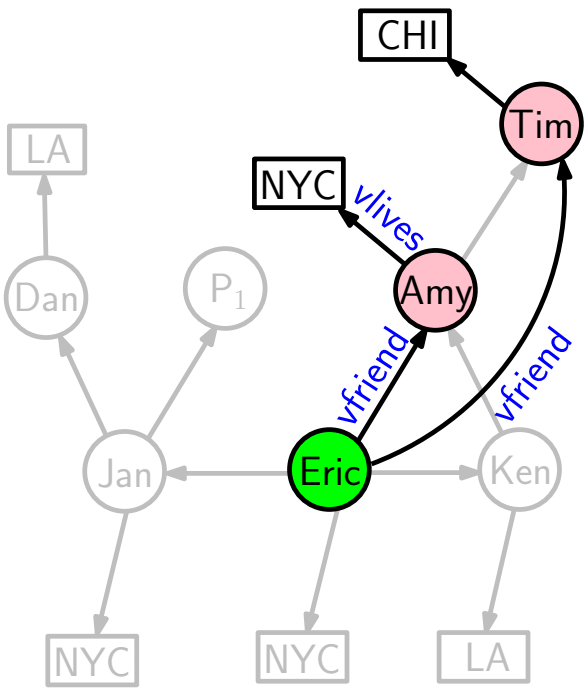
- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F:
CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

Introduction: A running example and solution outline

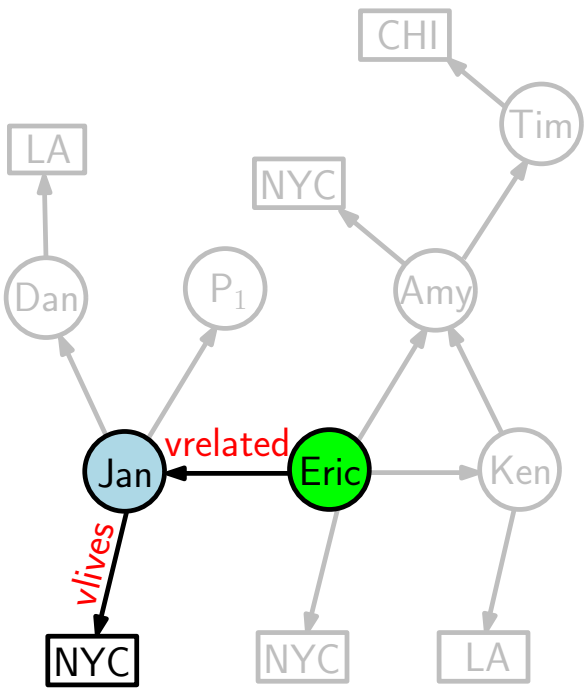
- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F:
CONSTRUCT{
?f$_0$ vfriend ?f$_1$,
?f$_1$ vlives ?l$_1$}
WHERE{
?f$_0$ name Eric,
?f$_0$ friend ?f$_1$,
?f$_1$ lives ?l$_1$}

V_{FoF}:
CONSTRUCT{
?f$_2$ vfriend ?f$_4$,
?f$_4$ vlives ?l$_4$}
WHERE{
?f$_2$ name Eric,
?f$_2$ friend ?f$_3$,
?f$_3$ friend ?f$_4$,
?f$_4$ lives ?l$_4$}

Introduction: A running example and solution outline



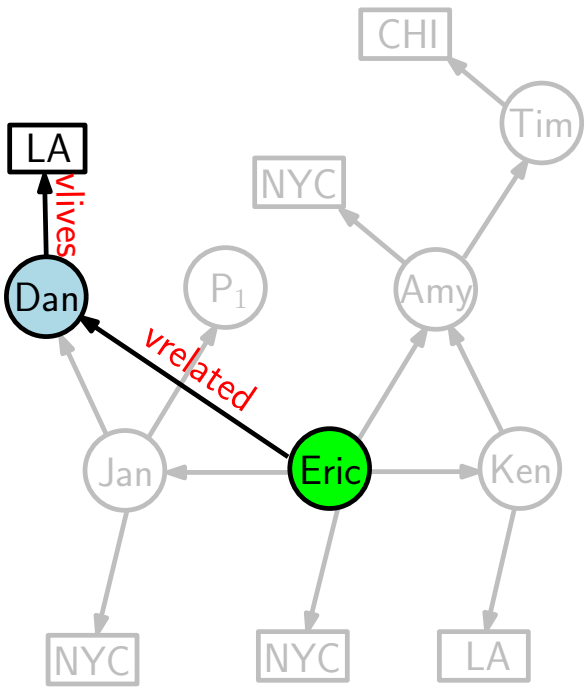
- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).

V_F:
CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
?f_2 vfriend ?f_4,
?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
?r_0 vrelated ?r_1,
?r_1 vlives ?l_1}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_1}

Introduction: A running example and solution outline



- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).

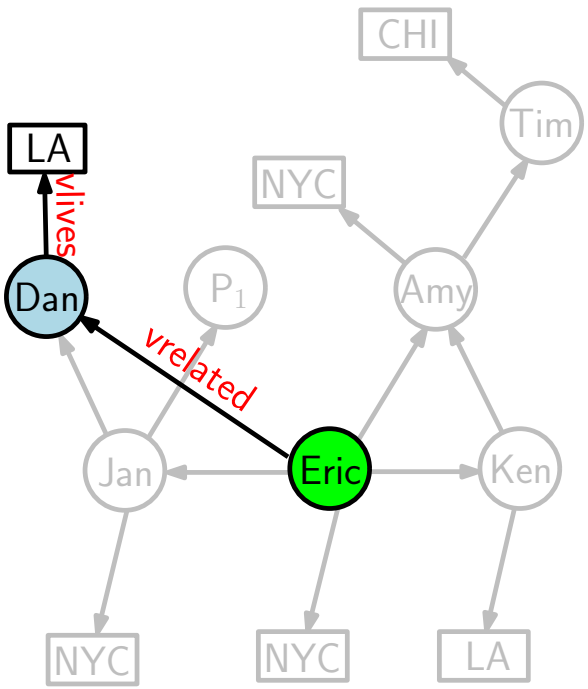
V_F:
CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
?f_2 vfriend ?f_4,
?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
?r_0 vrelated ?r_1,
?r_1 vlives ?l_1}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_1}

V_{RoR}:
CONSTRUCT{
?r_2 vrelated ?r_4,
?r_4 vlives ?l_4}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_4}

Introduction: A running example and solution outline



- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).

V_F:
CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

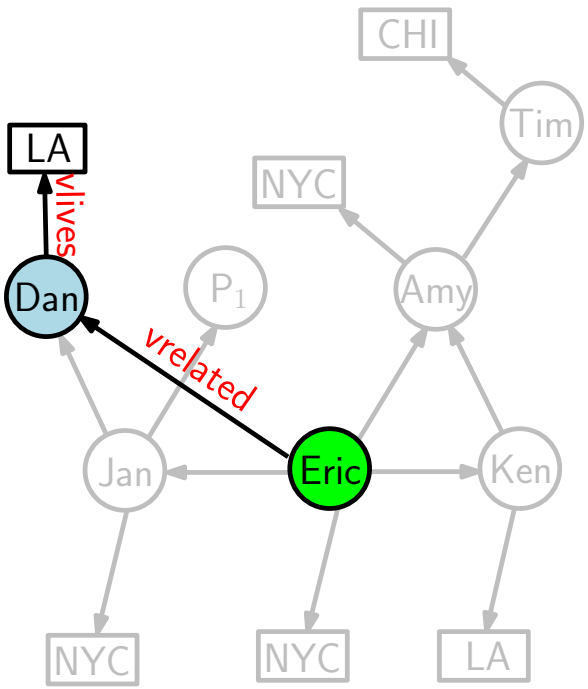
V_{FoF}:
CONSTRUCT{
?f_2 vfriend ?f_4,
?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
?r_0 vrelated ?r_1,
?r_1 vlives ?l_1}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_1}

V_{RoR}:
CONSTRUCT{
?r_2 vrelated ?r_4,
?r_4 vlives ?l_4}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_4}

Q_u:
SELECT ?f_5
WHERE {?p vfriend ?f_5 , ?f_5 vlives ?l_5,
?p vrelated ?r_5, ?r_5 vlives ?l_5.}

Introduction: A running example and solution outline



- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).

V_F:

CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:

CONSTRUCT{
?f_2 vfriend ?f_4,
?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:

CONSTRUCT{
?r_0 vrelated ?r_1,
?r_1 vlives ?l_1}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_1}

V_{RoR}:

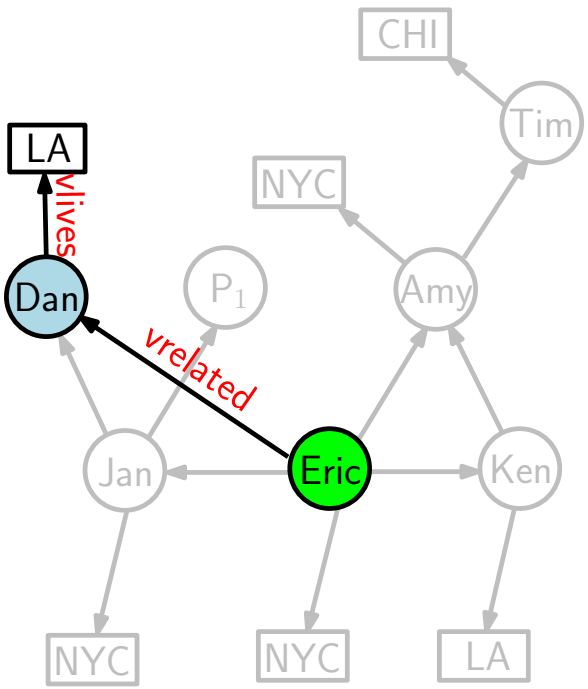
CONSTRUCT{
?r_2 vrelated ?r_4
?r_4 vlives ?l_4}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_4}

Q_u:

SELECT ?f_5
WHERE {?p vfriend ?f_5 , ?f_5 vlives ?l_5,
?p vrelated ?r_5, ?r_5 vlives ?l_5.}

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F:
CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
?f_2 vfriend ?f_4,
?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

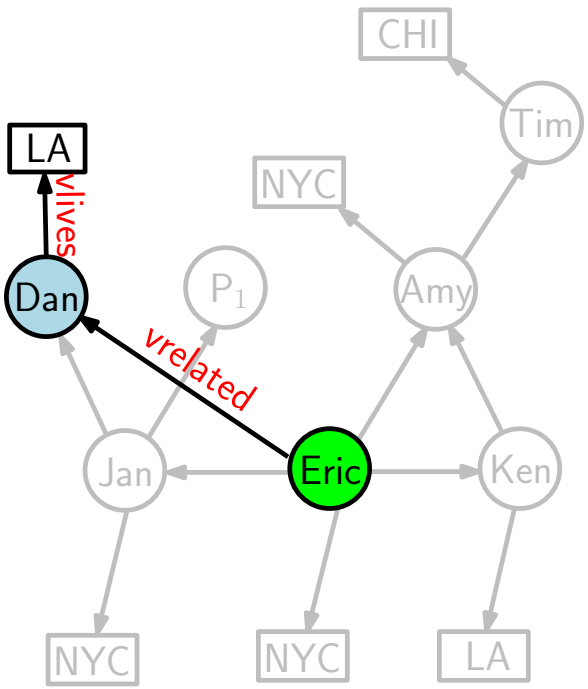
V_R:
CONSTRUCT{
?r_0 vrelated ?r_1,
?r_1 vlives ?l_1}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_1}

V_{RoR}:
CONSTRUCT{
?r_2 vrelated ?r_4
?r_4 vlives ?l_4}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_4}

Q_u:
SELECT ?f_5
WHERE {?p vfriend ?f_5 , ?f_5 vlives ?l_5,
?p vrelated ?r_5, ?r_5 vlives ?l_5.}

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F:
CONSTRUCT{
?f_0 vfriend ?f_1
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
?f_2 vfriend ?f_4,
?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

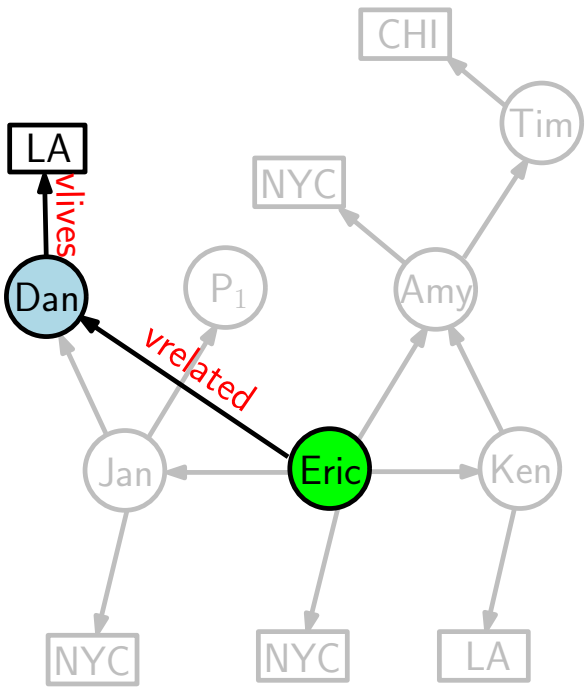
V_R:
CONSTRUCT{
?r_0 vrelated ?r_1,
?r_1 vlives ?l_1}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_1}

V_{RoR}:
CONSTRUCT{
?r_2 vrelated ?r_4
?r_4 vlives ?l_4}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_4}

Q_u:
SELECT ?f_5
WHERE {?p vfriend ?f_5 , ?f_5 vlives ?l_5,
?p vrelated ?r_5, ?r_5 vlives ?l_5.}

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F:
CONSTRUCT{
?f0 vfriend ?f1
?f1 vlives ?l1}
WHERE{
?f0 name Eric,
?f0 friend ?f1,
?f1 lives ?l1}

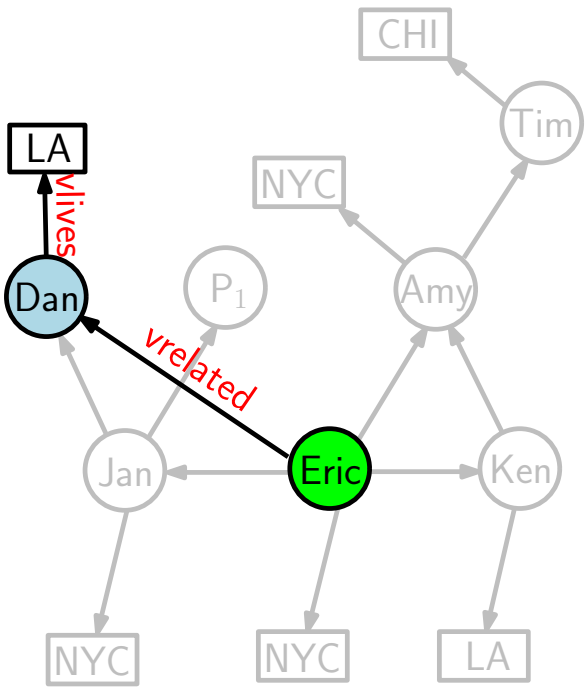
V_{FoF}:
CONSTRUCT{
?f2 vfriend ?f4,
?f4 vlives ?l4}
WHERE{
?f2 name Eric,
?f2 friend ?f3,
?f3 friend ?f4,
?f4 lives ?l4}

V_R:
CONSTRUCT{
?r0 vrelated ?r1,
?r1 vlives ?l1}
WHERE{
?r0 name Eric,
?r0 related ?r1,
?r1 lives ?l1}

V_{RoR}:
CONSTRUCT{
?r2 vrelated ?r4
?r4 vlives ?l4}
WHERE{
?r2 name Eric,
?r2 related ?r3,
?r3 related ?r4,
?r4 lives ?l4}

Q_u:
SELECT ?f5
WHERE {?p vfriend ?f5 , ?f5 vlives ?l5,
?p vrelated ?r5, ?r5 vlives ?l5.}

Introduction: A running example and solution outline



- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).

V_F:
CONSTRUCT{
?f_0 vfriend ?f_1,
?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
?f_2 vfriend ?f_4,
?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
?r_0 vrelated ?r_1,
?r_1 vlives ?l_1}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_1}

V_{RoR}:
CONSTRUCT{
?r_2 vrelated ?r_4,
?r_4 vlives ?l_4}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_4}

Q_u:
SELECT ?f_5
WHERE {?p vfriend ?f_5 , ?f_5 vlives ?l_5,
?p vrelated ?r_5, ?r_5 vlives ?l_5.}

Output exponential rewritings!

Outline

Outline

Rewriting Algorithm

Q_u:
SELECT ?f_5 WHERE{ ?p vfriend ?f_5 ?f_5 vlives ?l_5 ?p vrelated ?r_5 ?r_5 vlives ?l_5 }

V_F:
CONSTRUCT{
① ?f_0 vfriend ?f_1,
② ?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
① ?f_2 vfriend ?f_4,
② ?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
①?r_0 vrelated ?r_1,
②?r_1 vlives ?l_3}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_3}

V_{RoR}:
CONSTRUCT{
①?r_2 vrelated ?r_4,
②?r_4 vlives ?l_6}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_6}

Rewriting Algorithm

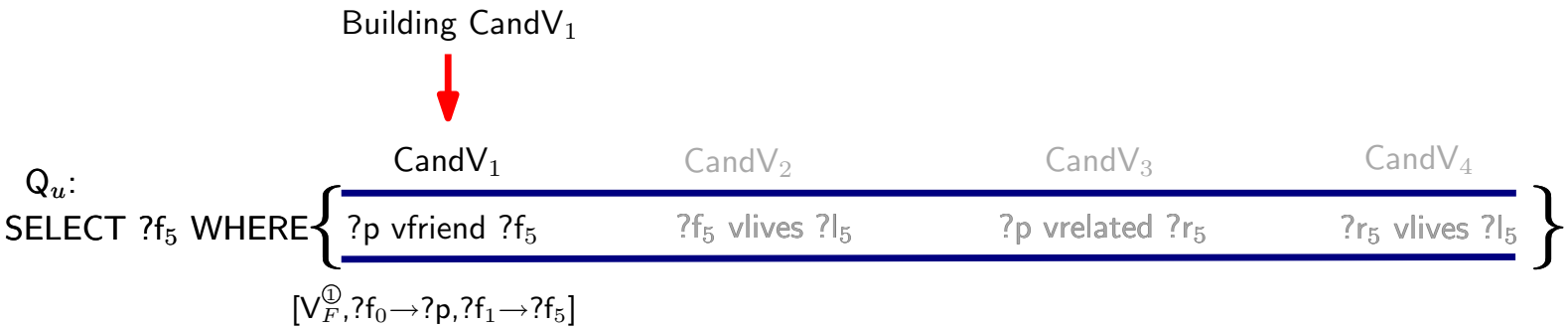
V_F:
CONSTRUCT{
① ?f_0 vfriend ?f_1,
② ?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
① ?f_2 vfriend ?f_4,
② ?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
①?r_0 vrelated ?r_1,
②?r_1 vlives ?l_3}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_3}

V_{RoR}:
CONSTRUCT{
①?r_2 vrelated ?r_4,
②?r_4 vlives ?l_6}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_6}

Rewriting Algorithm



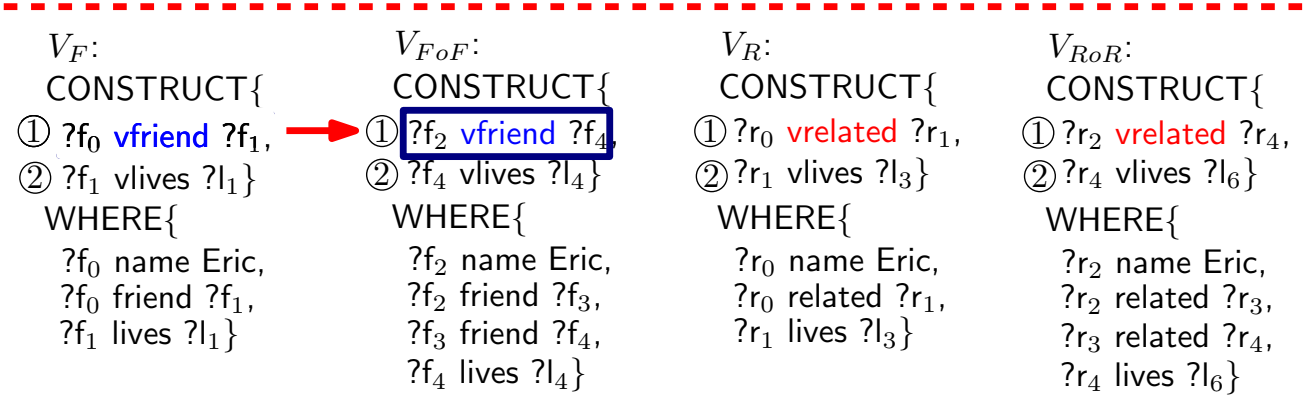
V_F:
CONSTRUCT{
① ?f0 vfriend ?f1,
② ?f1 vlives ?l1}
WHERE{
?f0 name Eric,
?f0 friend ?f1,
?f1 lives ?l1}

V_{FoF}:
CONSTRUCT{
① ?f2 vfriend ?f4,
② ?f4 vlives ?l4}
WHERE{
?f2 name Eric,
?f2 friend ?f3,
?f3 friend ?f4,
?f4 lives ?l4}

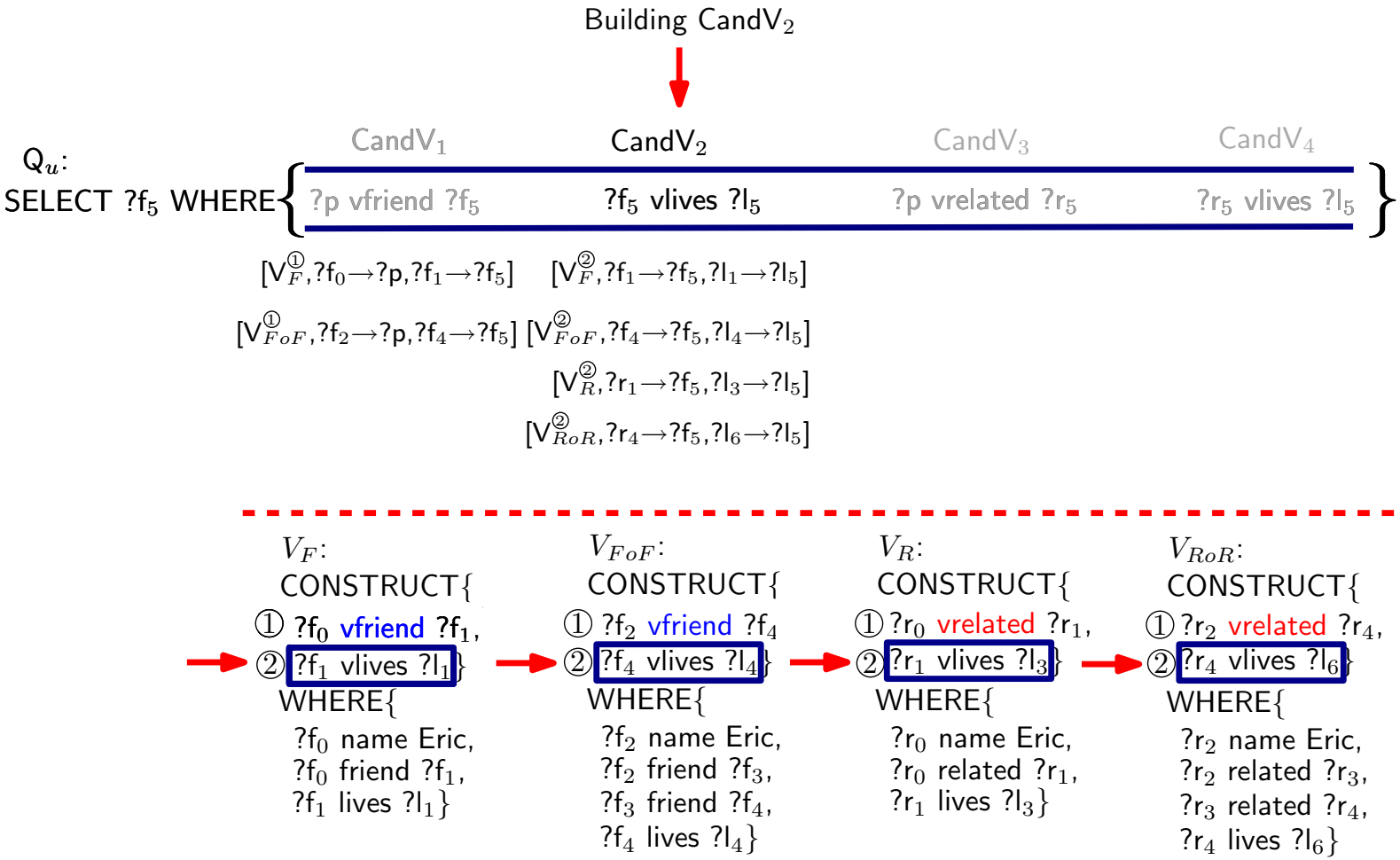
V_R:
CONSTRUCT{
① ?r0 vrelated ?r1,
② ?r1 vlives ?l3}
WHERE{
?r0 name Eric,
?r0 related ?r1,
?r1 lives ?l3}

V_{RoR}:
CONSTRUCT{
① ?r2 vrelated ?r4,
② ?r4 vlives ?l6}
WHERE{
?r2 name Eric,
?r2 related ?r3,
?r3 related ?r4,
?r4 lives ?l6}

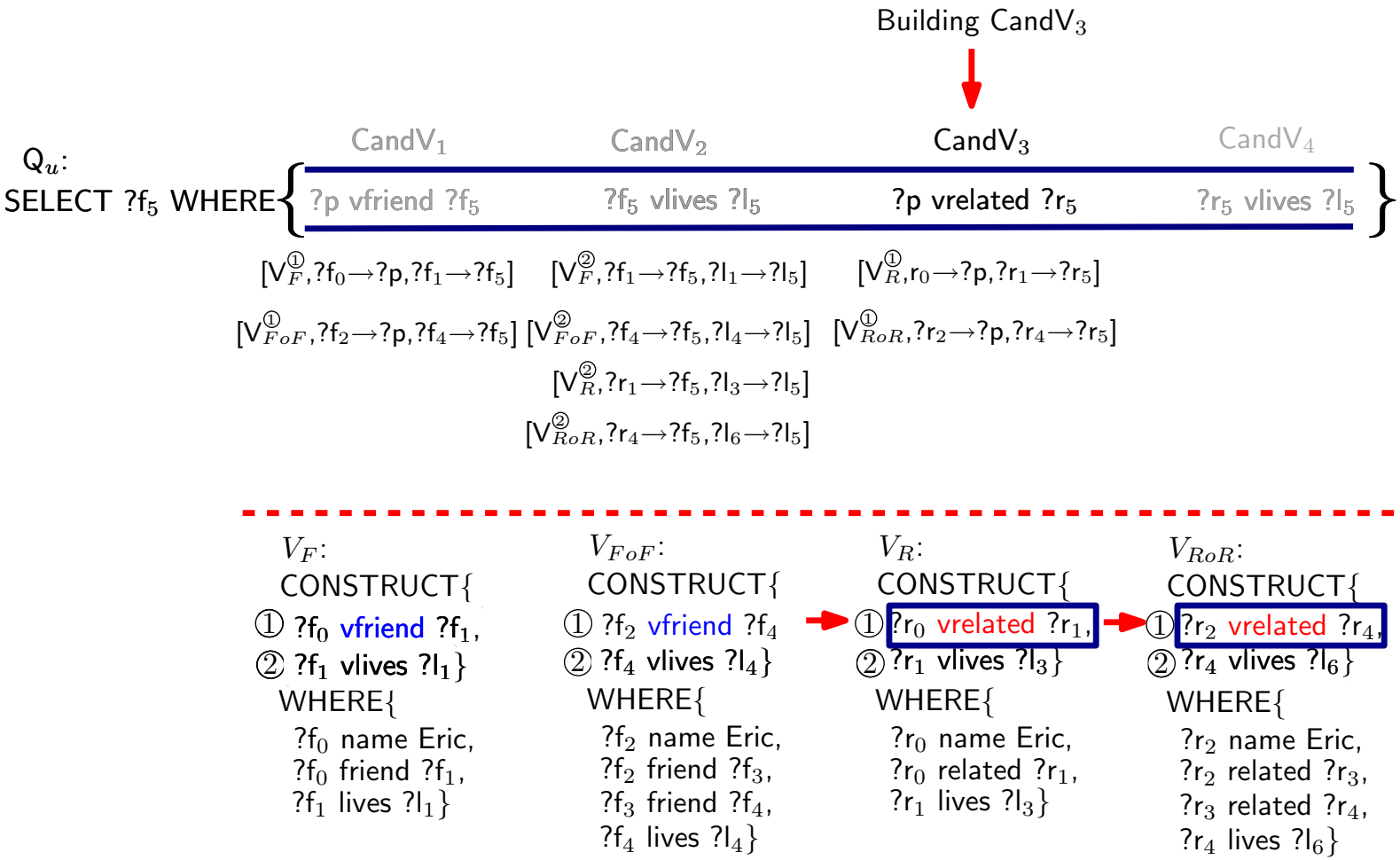
Rewriting Algorithm



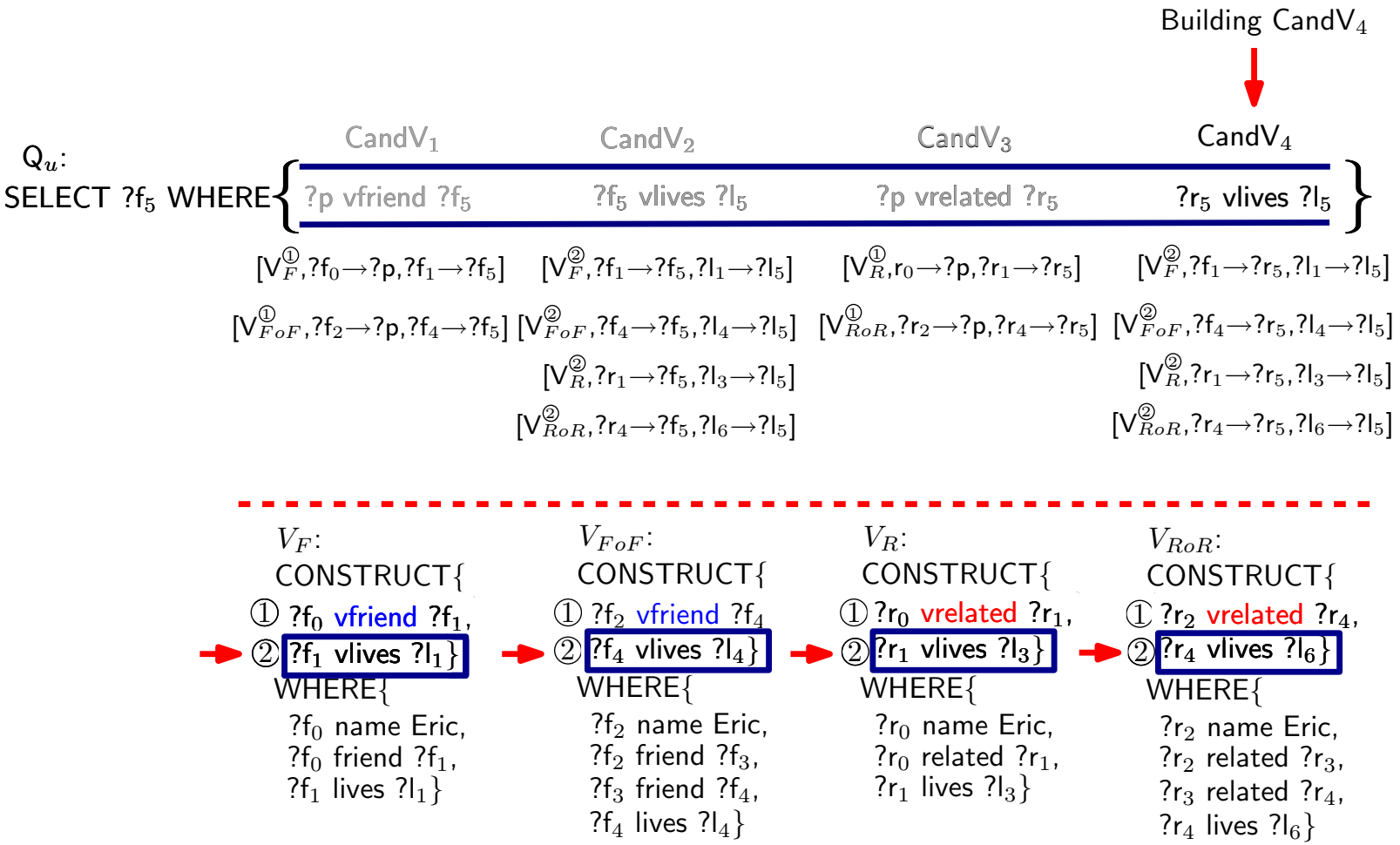
Rewriting Algorithm



Rewriting Algorithm



Rewriting Algorithm



Rewriting Algorithm

Q_u:
SELECT ?f_5 WHERE{

CandV$_1$	CandV$_2$	CandV$_3$	CandV$_4$
?p vfriend ?f_5	?f_5 vlives ?l_5	?p vrelated ?r_5	?r_5 vlives ?l_5
$[V_F^{①}$,?$f_0 \rightarrow$?p,?$f_1 \rightarrow$?$f_5]$	$[V_F^{②}$,?$f_1 \rightarrow$?f_5,?$l_1 \rightarrow$?$l_5]$	$[V_R^{①}$,$r_0 \rightarrow$?p,?$r_1 \rightarrow$?$r_5]$	$[V_F^{②}$,?$f_1 \rightarrow$?r_5,?$l_1 \rightarrow$?$l_5]$
$[V_{FoF}^{①}$,?$f_2 \rightarrow$?p,?$f_4 \rightarrow$?$f_5]$	$[V_{FoF}^{②}$,?$f_4 \rightarrow$?f_5,?$l_4 \rightarrow$?$l_5]$	$[V_{RoR}^{①}$,?$r_2 \rightarrow$?p,?$r_4 \rightarrow$?$r_5]$	$[V_{FoF}^{②}$,?$f_4 \rightarrow$?r_5,?$l_4 \rightarrow$?$l_5]$
	$[V_R^{②}$,?$r_1 \rightarrow$?f_5,?$l_3 \rightarrow$?$l_5]$		$[V_R^{②}$,?$r_1 \rightarrow$?r_5,?$l_3 \rightarrow$?$l_5]$
	$[V_{RoR}^{②}$,?$r_4 \rightarrow$?f_5,?$l_6 \rightarrow$?$l_5]$		$[V_{RoR}^{②}$,?$r_4 \rightarrow$?r_5,?$l_6 \rightarrow$?$l_5]$

V_F:
CONSTRUCT{
① ?f_0 vfriend ?f_1,
② ?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

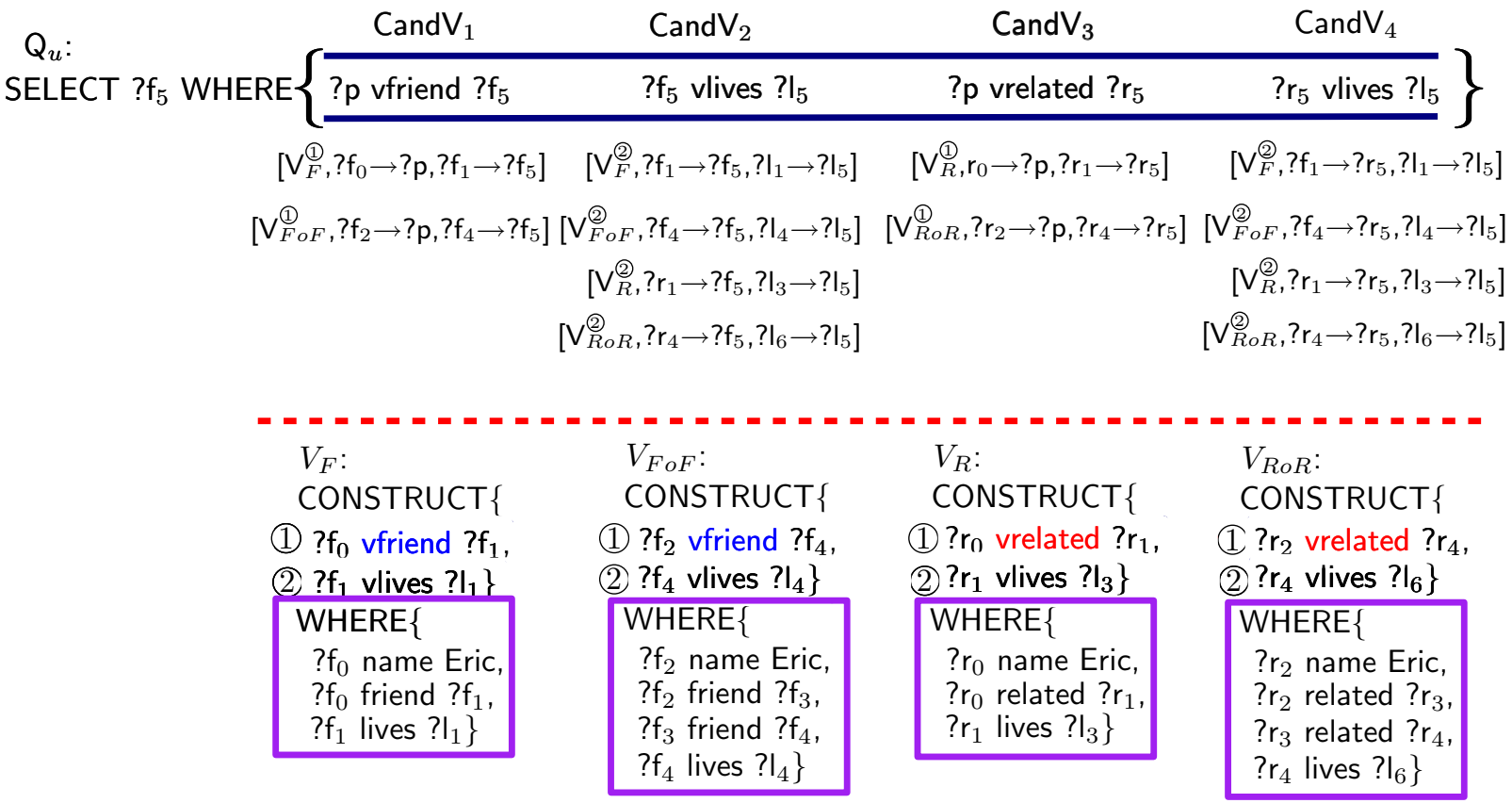
V_{FoF}:
CONSTRUCT{
① ?f_2 vfriend ?f_4,
② ?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
① ?r_0 vrelated ?r_1,
② ?r_1 vlives ?l_3}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_3}

V_{RoR}:
CONSTRUCT{
① ?r_2 vrelated ?r_4,
② ?r_4 vlives ?l_6}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_6}

Rewriting Algorithm

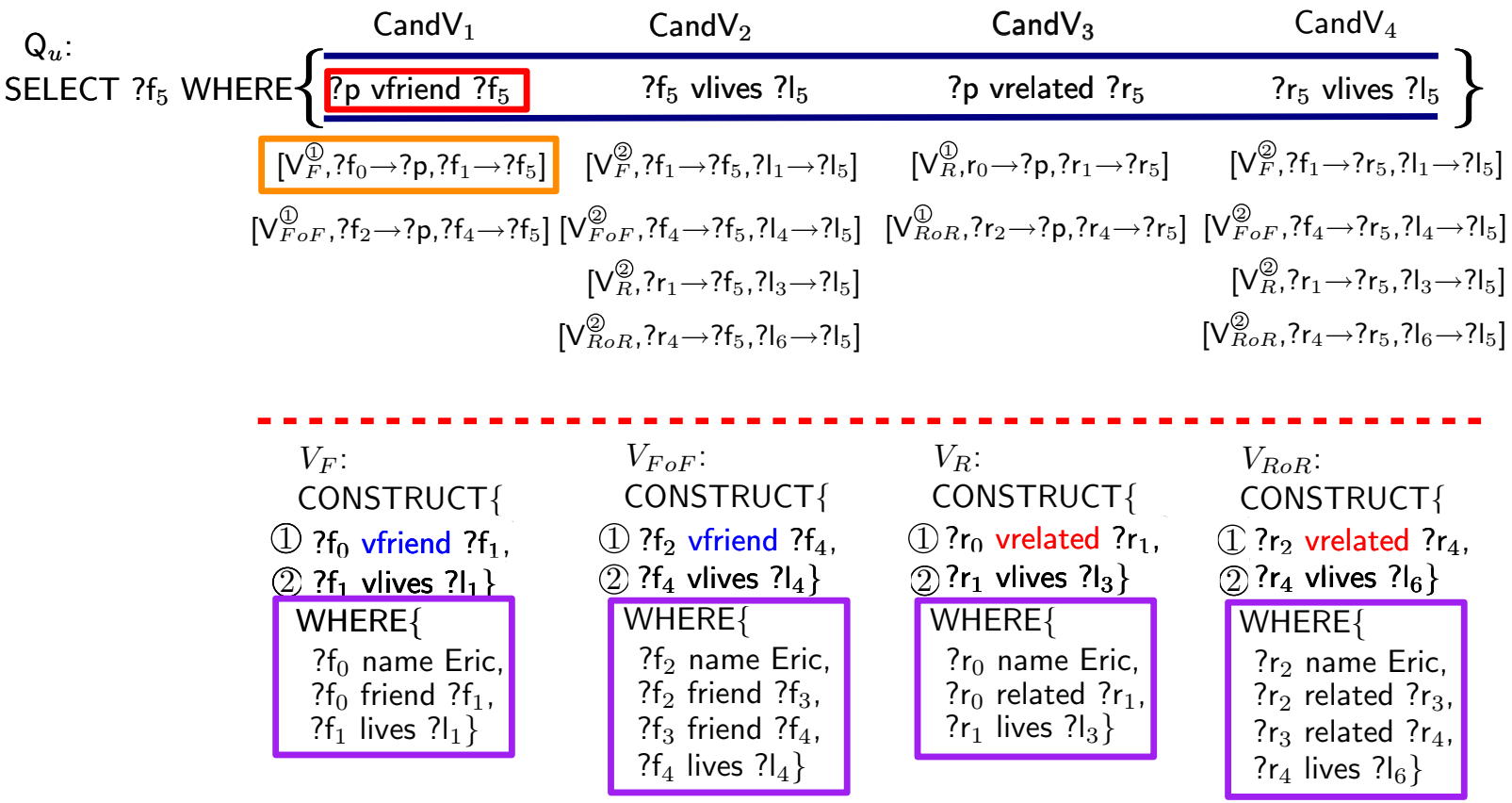
Generate one rewriting:



Rewriting Algorithm

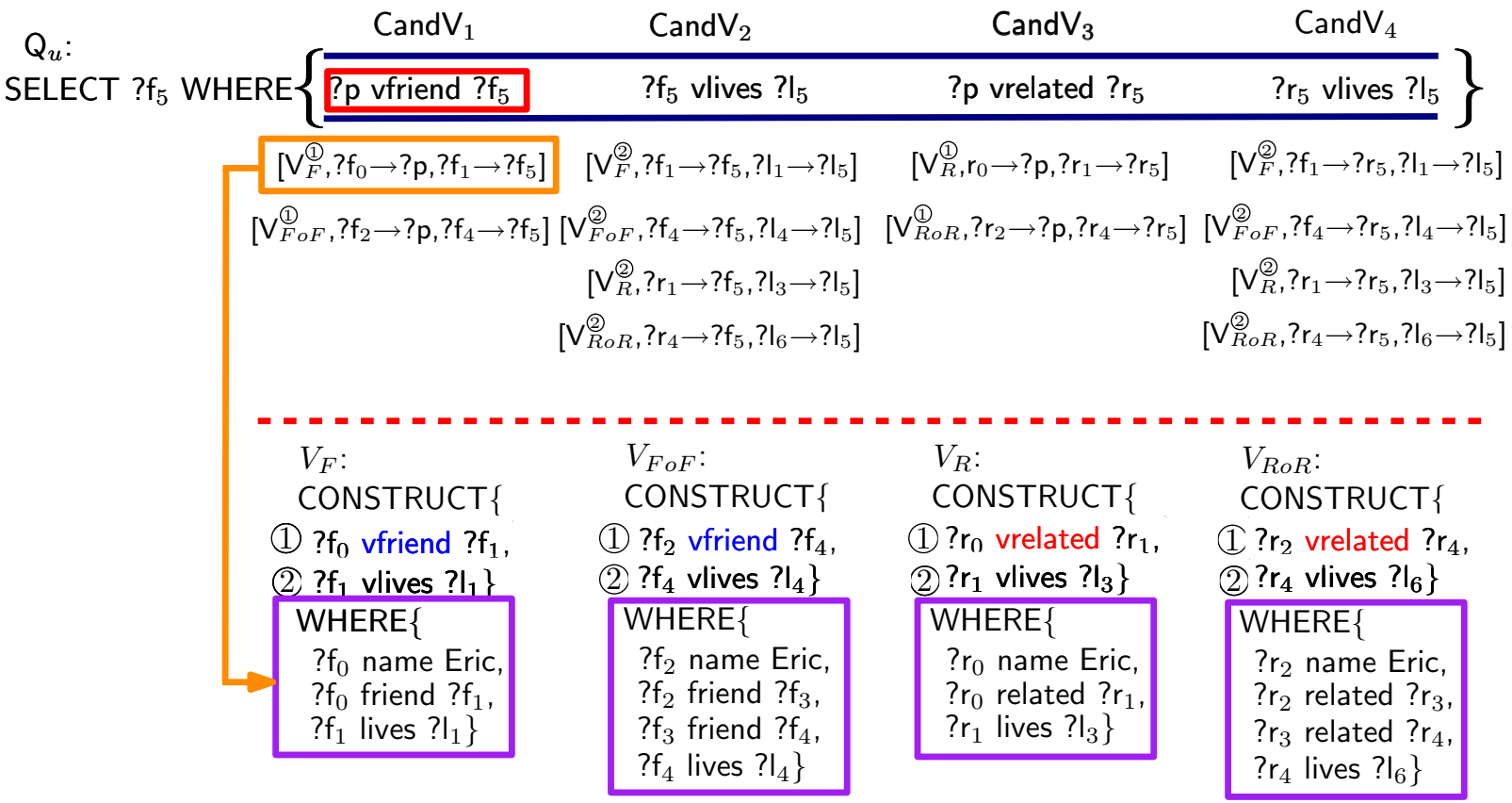
Generate one rewriting:

SELECT $?f_5$ WHERE {



Rewriting Algorithm

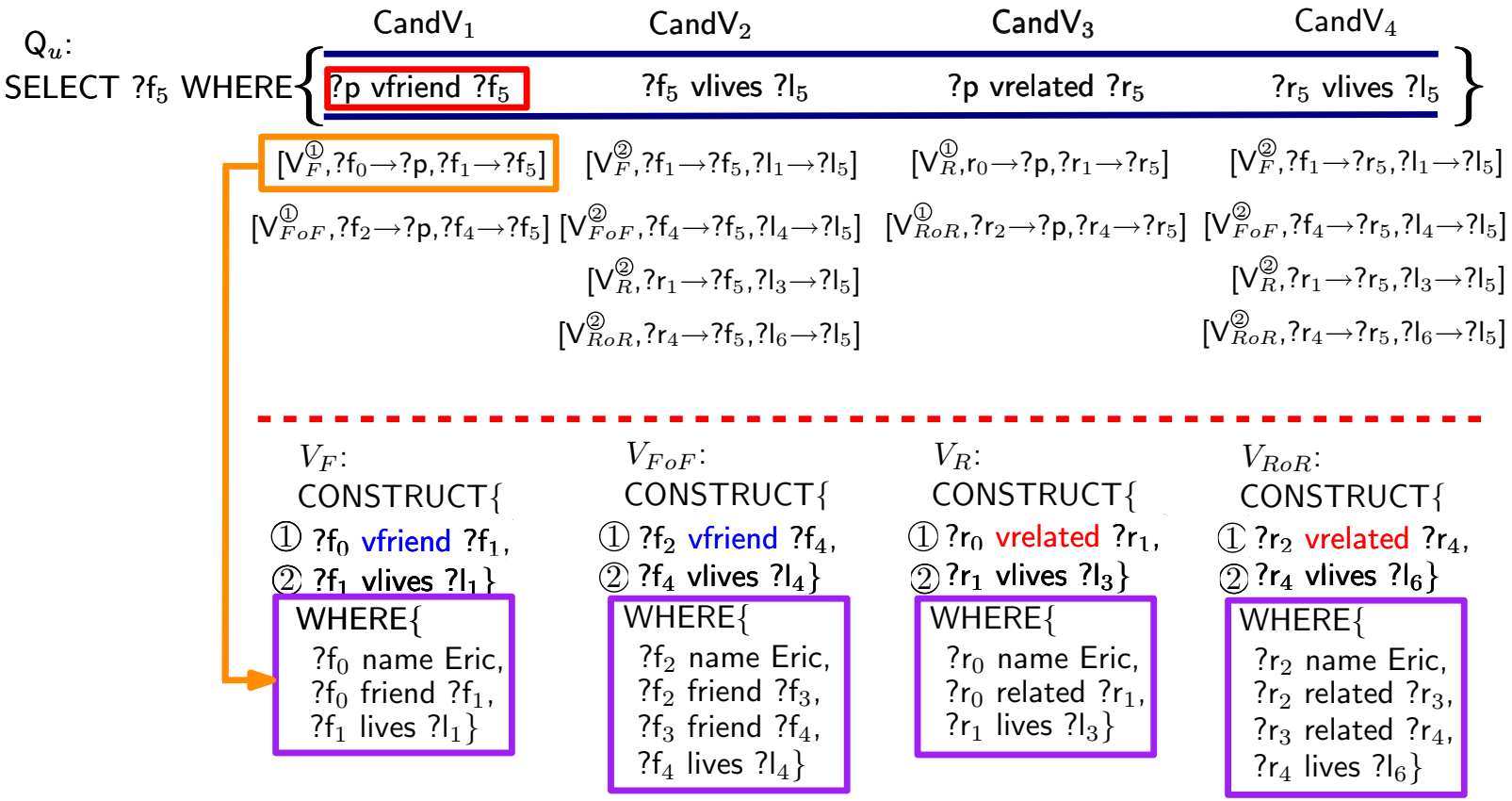
Generate one rewriting:
SELECT ?f_5 WHERE {



Rewriting Algorithm

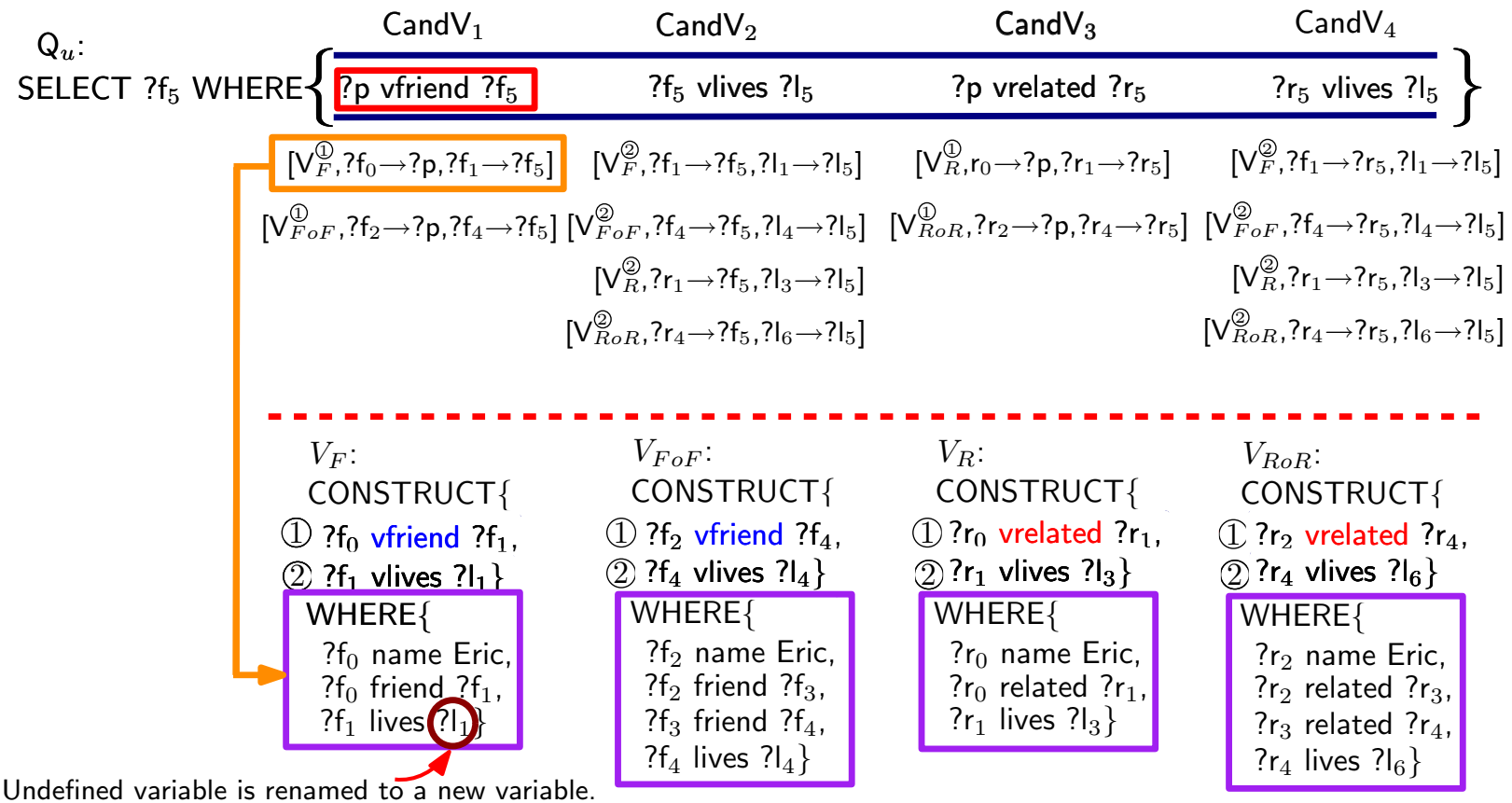
Generate one rewriting:

SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'$_1$,



Rewriting Algorithm

Generate one rewriting:
SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'_1,

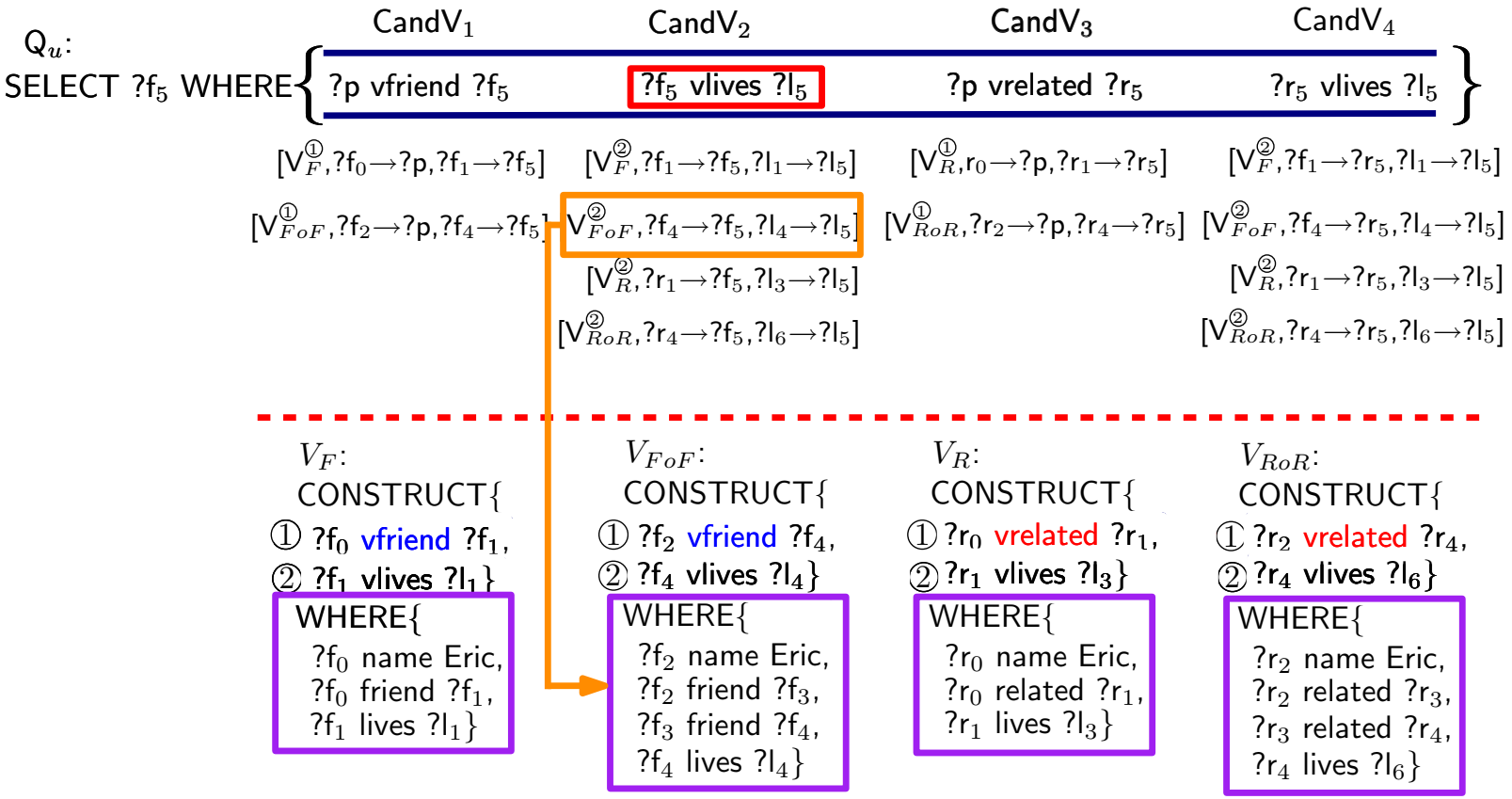


Undefined variable is renamed to a new variable.

Rewriting Algorithm

Generate one rewriting:

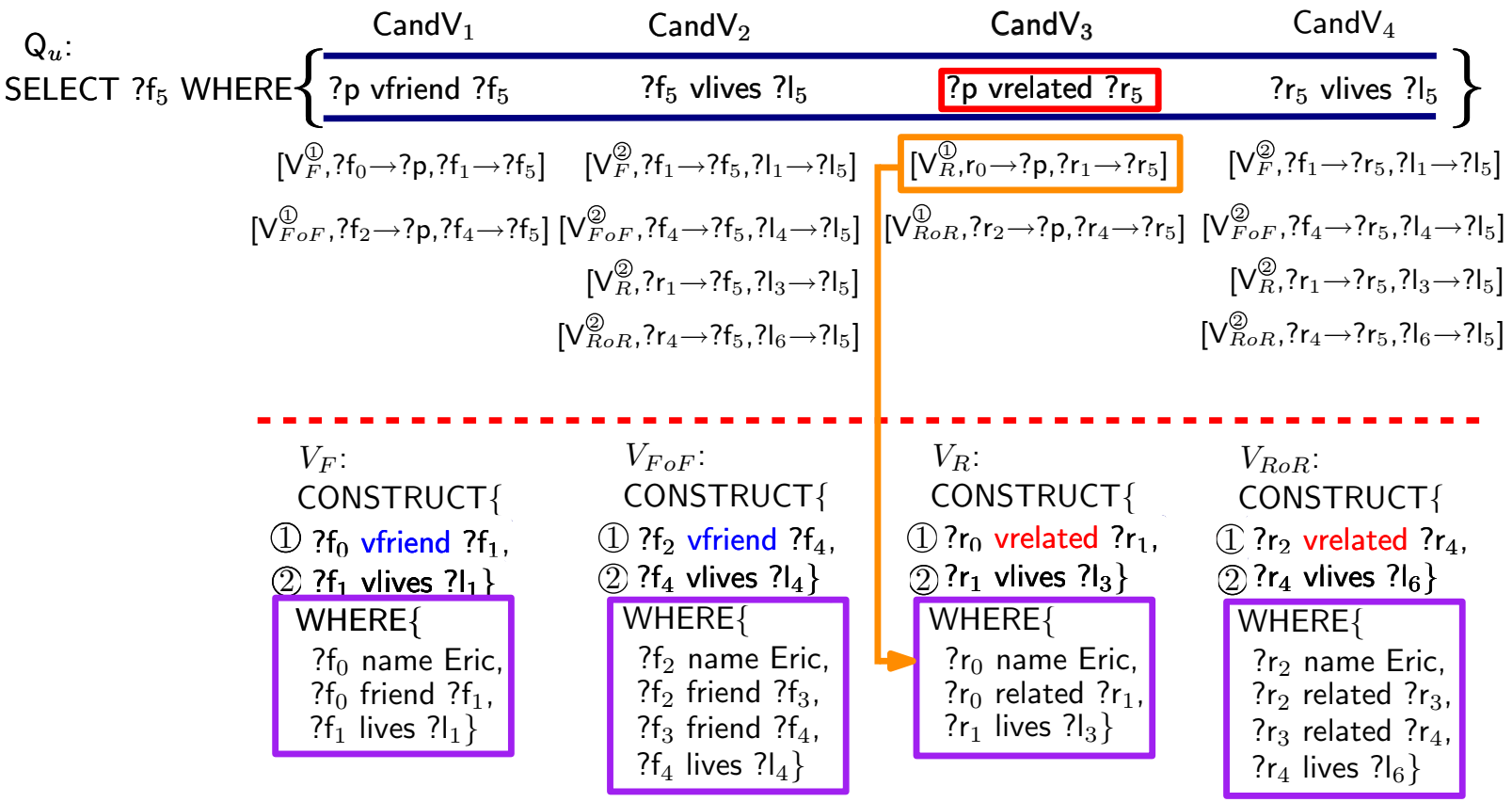
SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'_1,?f'_2 name Eric, ?f'_2 friend ?f'_3, ?f'_3 friend ?f_5, ?f_5 lives ?l_5



Rewriting Algorithm

Generate one rewriting:

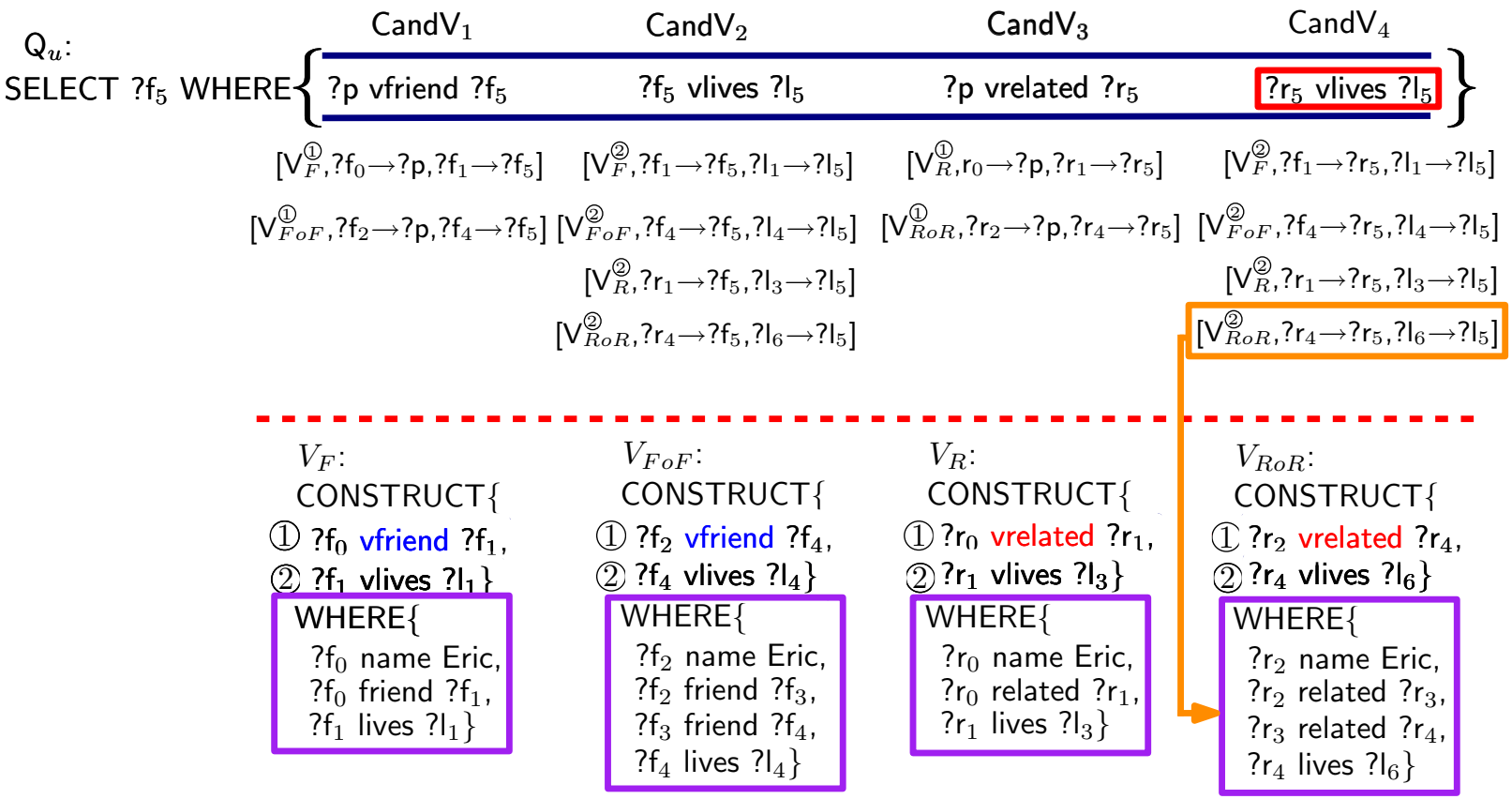
SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'_1,?f'_2 name Eric, ?f'_2 friend ?f'_3, ?f'_3 friend ?f_5, ?f_5 lives ?l_5
?p name Eric, ?p related ?r_5, ?r_5 lives ?l'_3,



Rewriting Algorithm

Generate one rewriting:

SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'_1,?f'_2 name Eric, ?f'_2 friend ?f'_3, ?f'_3 friend ?f_5, ?f_5 lives ?l_5
?p name Eric, ?p related ?r_5, ?r_5 lives ?l'_3,?r'_2 name Eric, ?r'_2 related ?r'_3, ?r'_3 related ?r_5, ?r_5 lives ?l_5}



Rewriting Algorithm

Generate one rewriting:

SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'_1,?f'_2 name Eric, ?f'_2 friend ?f'_3, ?f'_3 friend ?f_5, ?f_5 lives ?l_5
?p name Eric, ?p related ?r_5, ?r_5 lives ?l'_3,?r'_2 name Eric, ?r'_2 related ?r'_3, ?r'_3 related ?r_5, ?r_5 lives ?l_5}

Q_u:

	CandV$_1$	CandV$_2$	CandV$_3$	CandV$_4$
SELECT ?f_5 WHERE{	?p vfriend ?f_5	?f_5 vlives ?l_5	?p vrelated ?r_5	?r_5 vlives ?l_5 }
	$[V_F^{①}$,?$f_0 \to$?p,?$f_1 \to$?$f_5]$	$[V_F^{②}$,?$f_1 \to$?f_5,?$l_1 \to$?$l_5]$	$[V_R^{①}$,$r_0 \to$?p,?$r_1 \to$?$r_5]$	$[V_F^{②}$,?$f_1 \to$?r_5,?$l_1 \to$?$l_5]$
	$[V_{FoF}^{①}$,?$f_2 \to$?p,?$f_4 \to$?$f_5]$	$[V_{FoF}^{②}$,?$f_4 \to$?f_5,?$l_4 \to$?$l_5]$	$[V_{RoR}^{①}$,?$r_2 \to$?p,?$r_4 \to$?$r_5]$	$[V_{FoF}^{②}$,?$f_4 \to$?r_5,?$l_4 \to$?$l_5]$
		$[V_R^{②}$,?$r_1 \to$?f_5,?$l_3 \to$?$l_5]$		$[V_R^{②}$,?$r_1 \to$?r_5,?$l_3 \to$?$l_5]$
		$[V_{RoR}^{②}$,?$r_4 \to$?f_5,?$l_6 \to$?$l_5]$		$[V_{RoR}^{②}$,?$r_4 \to$?r_5,?$l_6 \to$?$l_5]$

V_F:
CONSTRUCT{
① ?f_0 vfriend ?f_1,
② ?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
① ?f_2 vfriend ?f_4,
② ?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
① ?r_0 vrelated ?r_1,
② ?r_1 vlives ?l_3}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_3}

V_{RoR}:
CONSTRUCT{
① ?r_2 vrelated ?r_4,
② ?r_4 vlives ?l_6}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_6}

- There are totally |CandV$_1$|×|CandV$_2$|×|CandV$_3$|×|CandV$_4$| rewritings modulos a compatible check.

Rewriting Algorithm

Generate one rewriting:

SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'_1,?f'_2 name Eric, ?f'_2 friend ?f'_3, ?f'_3 friend ?f_5, ?f_5 lives ?l_5
?p name Eric, ?p related ?r_5, ?r_5 lives ?l'_3,?r'_2 name Eric, ?r'_2 related ?r'_3, ?r'_3 related ?r_5, ?r_5 lives ?l_5}

Q_u:
SELECT ?f_5 WHERE{

CandV$_1$	CandV$_2$	CandV$_3$	CandV$_4$
?p vfriend ?f_5	?f_5 vlives ?l_5	?p vrelated ?r_5	?r_5 vlives ?l_5
$[V_F^{①},?f_0\to?p,?f_1\to?f_5]$	$[V_F^{②},?f_1\to?f_5,?l_1\to?l_5]$	$[V_R^{①},r_0\to?p,?r_1\to?r_5]$	$[V_F^{②},?f_1\to?r_5,?l_1\to?l_5]$
$[V_{FoF}^{①},?f_2\to?p,?f_4\to?f_5]$	$[V_{FoF}^{②},?f_4\to?f_5,?l_4\to?l_5]$	$[V_{RoR}^{①},?r_2\to?p,?r_4\to?r_5]$	$[V_{FoF}^{②},?f_4\to?r_5,?l_4\to?l_5]$
	$[V_R^{②},?r_1\to?f_5,?l_3\to?l_5]$		$[V_R^{②},?r_1\to?r_5,?l_3\to?l_5]$
	$[V_{RoR}^{②},?r_4\to?f_5,?l_6\to?l_5]$		$[V_{RoR}^{②},?r_4\to?r_5,?l_6\to?l_5]$

}

V_F:
CONSTRUCT{
① ?f_0 vfriend ?f_1,
② ?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
① ?f_2 vfriend ?f_4,
② ?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
① ?r_0 vrelated ?r_1,
② ?r_1 vlives ?l_3}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_3}

V_{RoR}:
CONSTRUCT{
① ?r_2 vrelated ?r_4,
② ?r_4 vlives ?l_6}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_6}

- There are totally |CandV$_1$|×|CandV$_2$|×|CandV$_3$|×|CandV$_4$| rewritings modulos a compatible check.
- CandVs are considered *incompatible* if they map the same variable to different constants.

Outline

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

The set of rewritten SPARQL queries is sound and complete.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

The set of rewritten SPARQL queries is sound and complete.

- So, there is no redundancy nor misses resulted from the rewriting.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

The set of rewritten SPARQL queries is sound and complete.

- So, there is no redundancy nor misses resulted from the rewriting.
- So, doing rewriting in SQL will not give less rewritings as the set of semantic units is tight already.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

The set of rewritten SPARQL queries is sound and complete.

- So, there is no redundancy nor misses resulted from the rewriting.
- So, doing rewriting in SQL will not give less rewritings as the set of semantic units is tight already.
- In the experiment, we show that SQL translation is indeed helpless.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

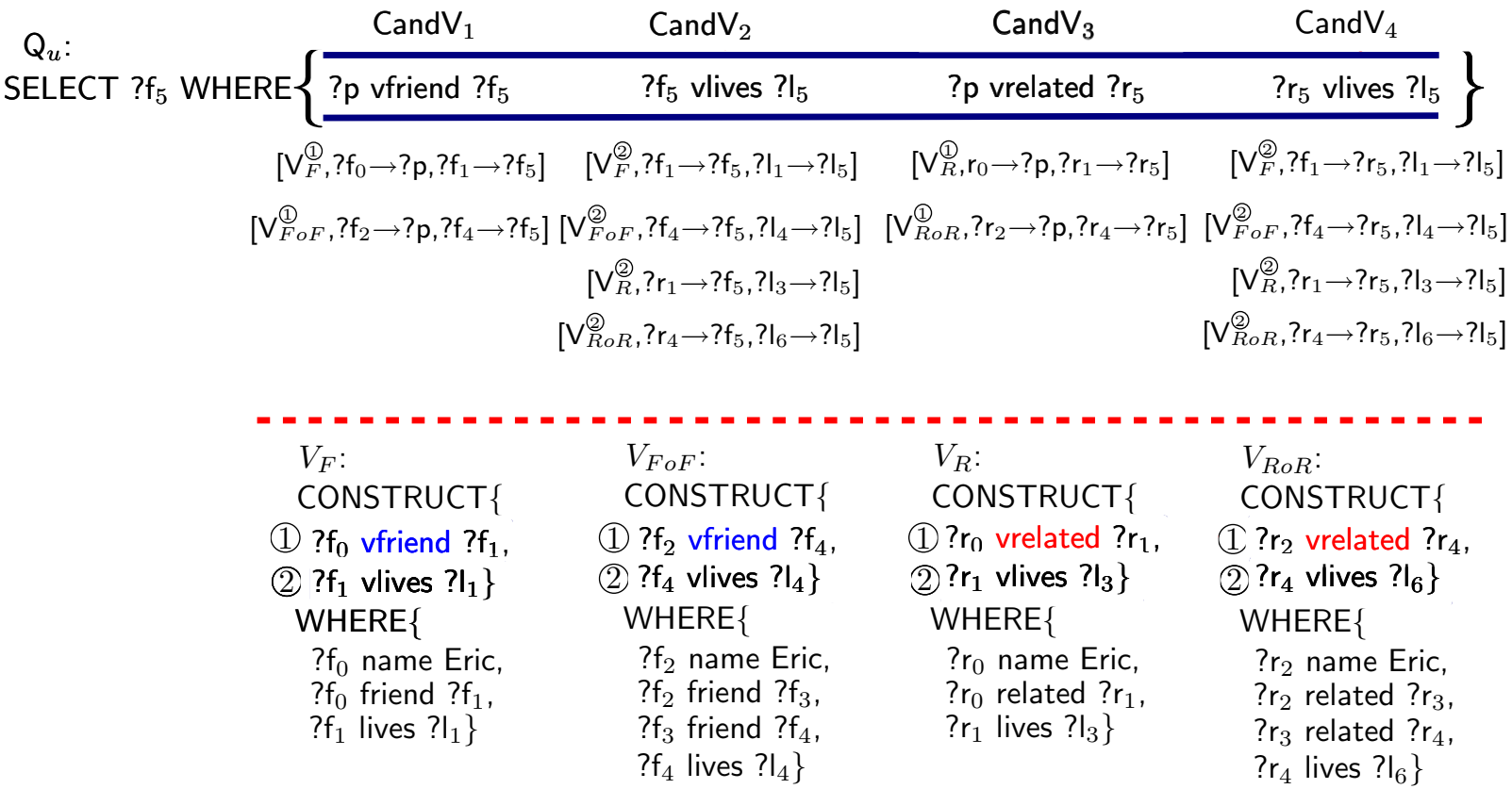
The set of rewritten SPARQL queries is sound and complete.

- So, there is no redundancy nor misses resulted from the rewriting.
- So, doing rewriting in SQL will not give less rewritings as the set of semantic units is tight already.
- In the experiment, we show that SQL translation is indeed helpless.
- But still, can we do better?

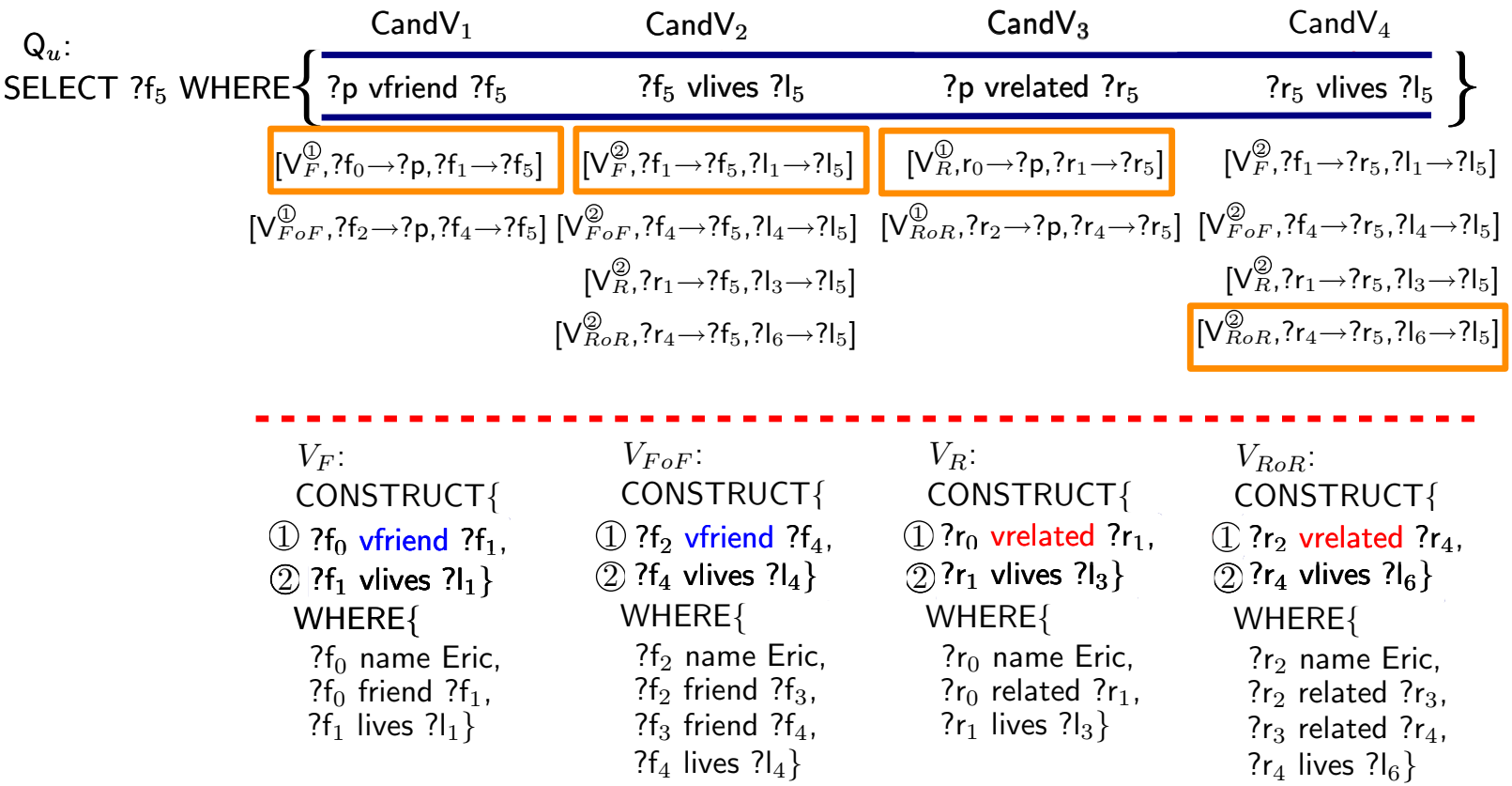
Outline

Outline

Optimization: minimize a rewritten query



Optimization: minimize a rewritten query



Optimization: minimize a rewritten query

Rewriting q:

SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'$_1$, ?f'$_0$ name Eric, ?f'$_0$ friend ?f_5, ?f_5 lives ?l_5,
?p name Eric, ?p related ?r_5, ?r_5 lives ?l'$_3$,?r'$_2$ name Eric, ?r'$_2$ related ?r'$_3$, ?r'$_3$ related ?r_5, ?r_5 lives ?l_5}

Q_u:
SELECT ?f_5 WHERE{ ?p vfriend ?f_5 | ?f_5 vlives ?l_5 | ?p vrelated ?r_5 | ?r_5 vlives ?l_5 }

CandV$_1$	CandV$_2$	CandV$_3$	CandV$_4$
[$V_F^{①}$,?f_0→?p,?f_1→?f_5]	[$V_F^{②}$,?f_1→?f_5,?l_1→?l_5]	[$V_R^{①}$,r_0→?p,?r_1→?r_5]	[$V_F^{②}$,?f_1→?r_5,?l_1→?l_5]
[$V_{FoF}^{①}$,?f_2→?p,?f_4→?f_5]	[$V_{FoF}^{②}$,?f_4→?f_5,?l_4→?l_5]	[$V_{RoR}^{①}$,?r_2→?p,?r_4→?r_5]	[$V_{FoF}^{②}$,?f_4→?r_5,?l_4→?l_5]
	[$V_R^{②}$,?r_1→?f_5,?l_3→?l_5]		[$V_R^{②}$,?r_1→?r_5,?l_3→?l_5]
	[$V_{RoR}^{②}$,?r_4→?f_5,?l_6→?l_5]		[$V_{RoR}^{②}$,?r_4→?r_5,?l_6→?l_5]

V_F:
CONSTRUCT{
① ?f_0 vfriend ?f_1,
② ?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
① ?f_2 vfriend ?f_4,
② ?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
① ?r_0 vrelated ?r_1,
② ?r_1 vlives ?l_3}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_3}

V_{RoR}:
CONSTRUCT{
① ?r_2 vrelated ?r_4,
② ?r_4 vlives ?l_6}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_6}

Optimization: minimize a rewritten query

Rewriting q:

SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'$_1$, ?f'$_0$ name Eric, ?f'$_0$ friend ?f_5, ?f_5 lives ?l_5,
?p name Eric, ?p related ?r_5, ?r_5 lives ?l'$_3$,?r'$_2$ name Eric, ?r'$_2$ related ?r'$_3$, ?r'$_3$ related ?r_5, ?r_5 lives ?l_5}

Q_u:
SELECT ?f_5 WHERE

CandV$_1$	CandV$_2$	CandV$_3$	CandV$_4$
?p vfriend ?f_5	?f_5 vlives ?l_5	?p vrelated ?r_5	?r_5 vlives ?l_5
$[V_F^{①}, ?f_0 \rightarrow ?p, ?f_1 \rightarrow ?f_5]$	$[V_F^{②}, ?f_1 \rightarrow ?f_5, ?l_1 \rightarrow ?l_5]$	$[V_R^{①}, ?r_0 \rightarrow ?p, ?r_1 \rightarrow ?r_5]$	$[V_F^{②}, ?f_1 \rightarrow ?r_5, ?l_1 \rightarrow ?l_5]$
$[V_{FoF}^{①}, ?f_2 \rightarrow ?p, ?f_4 \rightarrow ?f_5]$	$[V_{FoF}^{②}, ?f_4 \rightarrow ?f_5, ?l_4 \rightarrow ?l_5]$	$[V_{RoR}^{①}, ?r_2 \rightarrow ?p, ?r_4 \rightarrow ?r_5]$	$[V_{FoF}^{②}, ?f_4 \rightarrow ?r_5, ?l_4 \rightarrow ?l_5]$
	$[V_R^{②}, ?r_1 \rightarrow ?f_5, ?l_3 \rightarrow ?l_5]$		$[V_R^{②}, ?r_1 \rightarrow ?r_5, ?l_3 \rightarrow ?l_5]$
	$[V_{RoR}^{②}, ?r_4 \rightarrow ?f_5, ?l_6 \rightarrow ?l_5]$		$[V_{RoR}^{②}, ?r_4 \rightarrow ?r_5, ?l_6 \rightarrow ?l_5]$

V_F:
CONSTRUCT{
① ?f_0 vfriend ?f_1,
② ?f_1 vlives ?l_1}
WHERE{
?f_0 name Eric,
?f_0 friend ?f_1,
?f_1 lives ?l_1}

V_{FoF}:
CONSTRUCT{
① ?f_2 vfriend ?f_4,
② ?f_4 vlives ?l_4}
WHERE{
?f_2 name Eric,
?f_2 friend ?f_3,
?f_3 friend ?f_4,
?f_4 lives ?l_4}

V_R:
CONSTRUCT{
① ?r_0 vrelated ?r_1,
② ?r_1 vlives ?l_3}
WHERE{
?r_0 name Eric,
?r_0 related ?r_1,
?r_1 lives ?l_3}

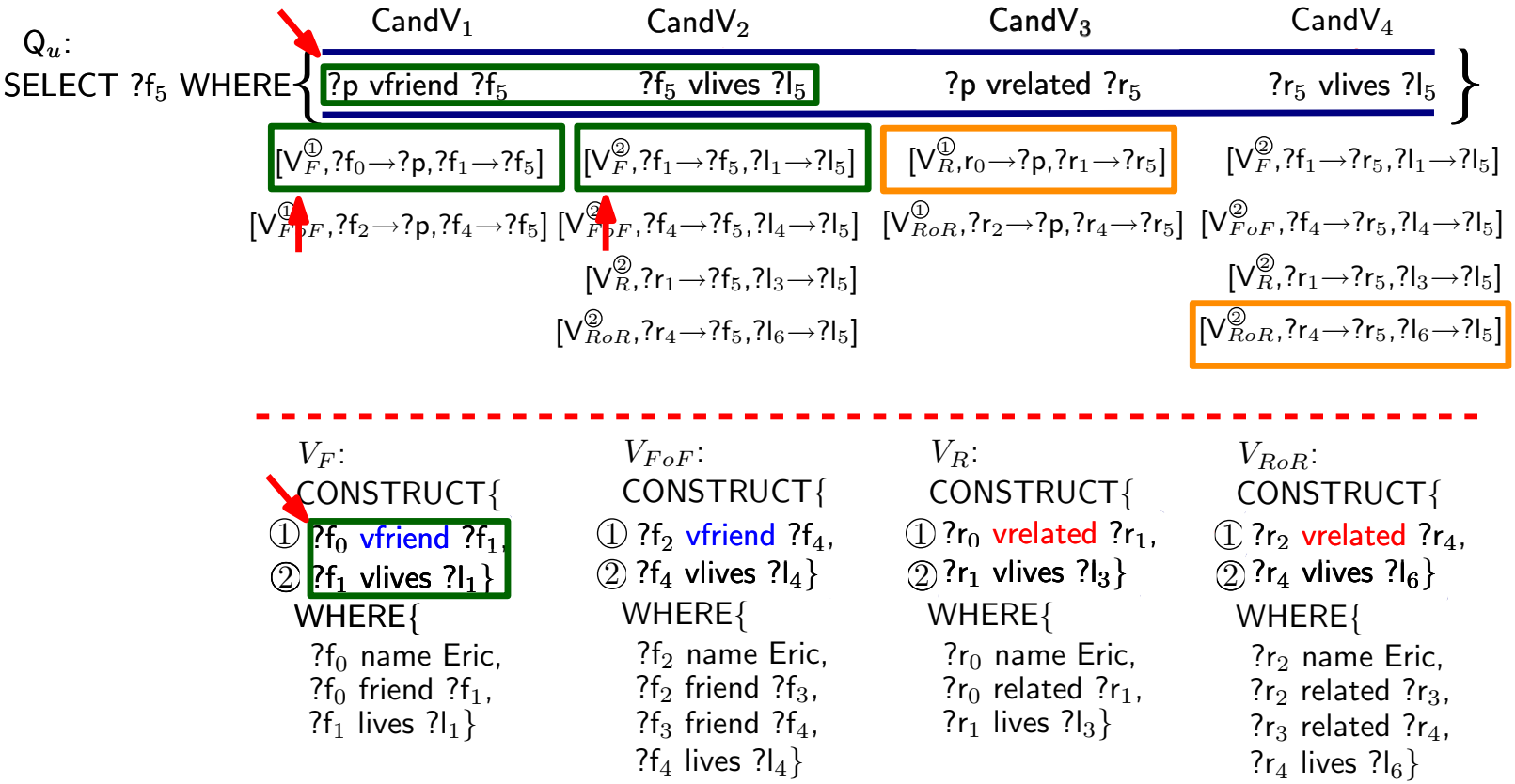
V_{RoR}:
CONSTRUCT{
① ?r_2 vrelated ?r_4,
② ?r_4 vlives ?l_6}
WHERE{
?r_2 name Eric,
?r_2 related ?r_3,
?r_3 related ?r_4,
?r_4 lives ?l_6}

- From the same view.

Optimization: minimize a rewritten query

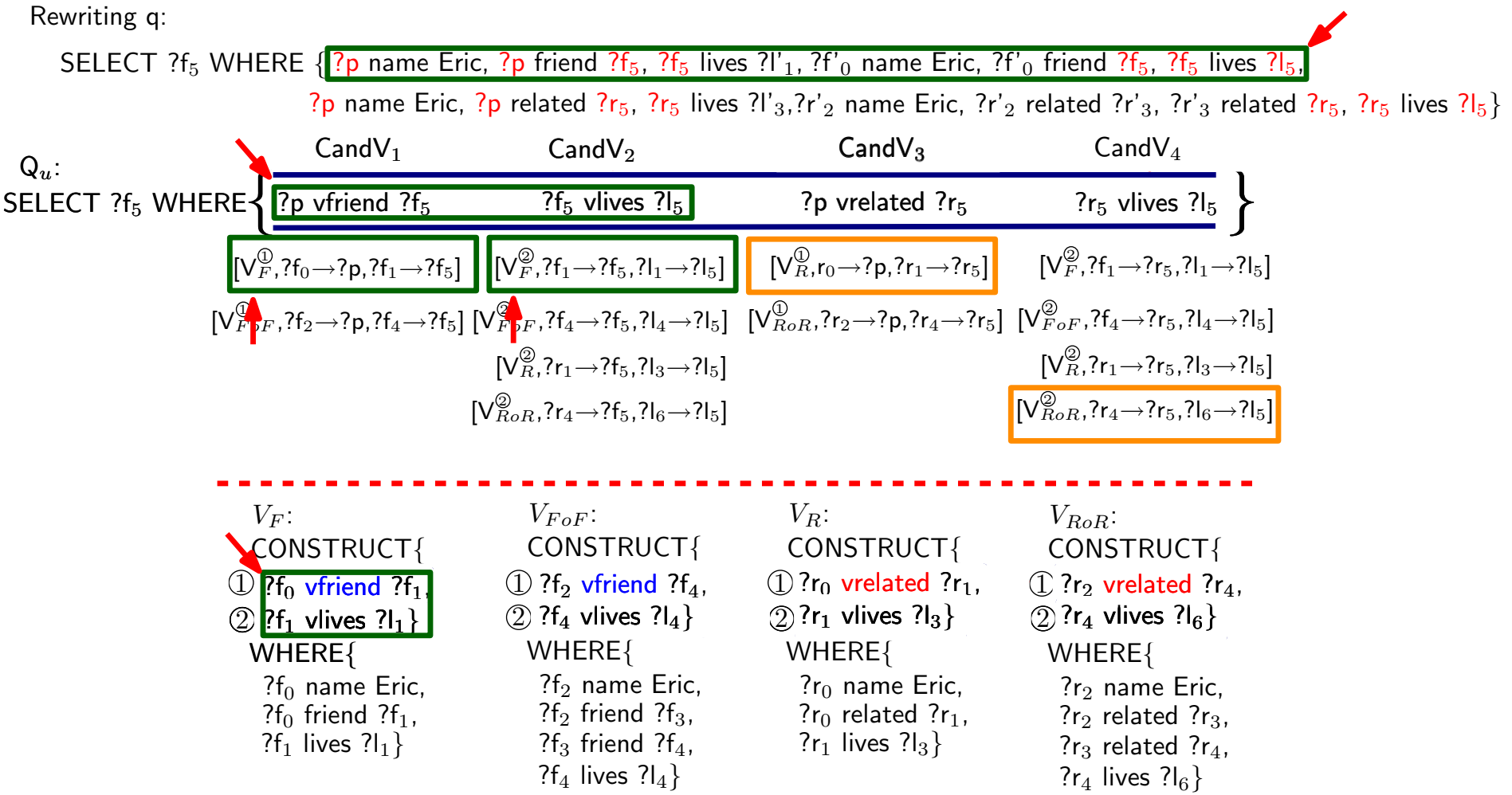
Rewriting q:

SELECT ?f_5 WHERE { ?p name Eric, ?p friend ?f_5, ?f_5 lives ?l'$_1$, ?f'$_0$ name Eric, ?f'$_0$ friend ?f_5, ?f_5 lives ?l_5,
?p name Eric, ?p related ?r_5, ?r_5 lives ?l'$_3$,?r'$_2$ name Eric, ?r'$_2$ related ?r'$_3$, ?r'$_3$ related ?r_5, ?r_5 lives ?l_5}



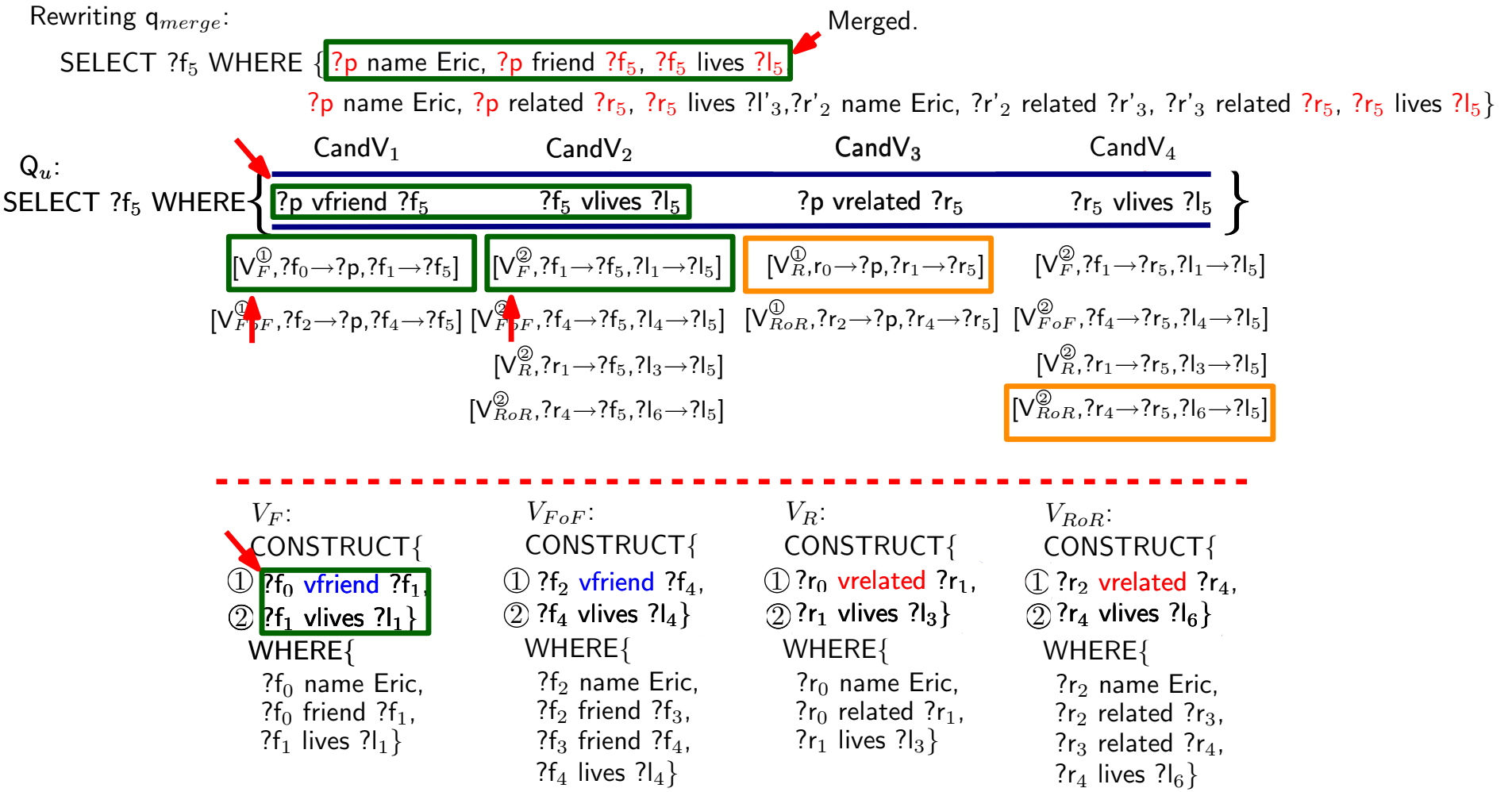
- From the same view.
- Join in the same way in query and view.

Optimization: minimize a rewritten query



- From the same view.
- Join in the same way in query and view.
- Can be merged.

Optimization: minimize a rewritten query



- From the same view.
- Join in the same way in query and view.
- Can be merged.

Optimization: minimize a rewritten query

- The proposed unification could potentially lead to a more efficient evaluation of the query.

Optimization: minimize a rewritten query

- The proposed unification could potentially lead to a more efficient evaluation of the query.
- We also show that such optimization will not affect the semantics of the query (sound and complete).

Optimization: minimize a rewritten query

- The proposed unification could potentially lead to a more efficient evaluation of the query.
- We also show that such optimization will not affect the semantics of the query (sound and complete).

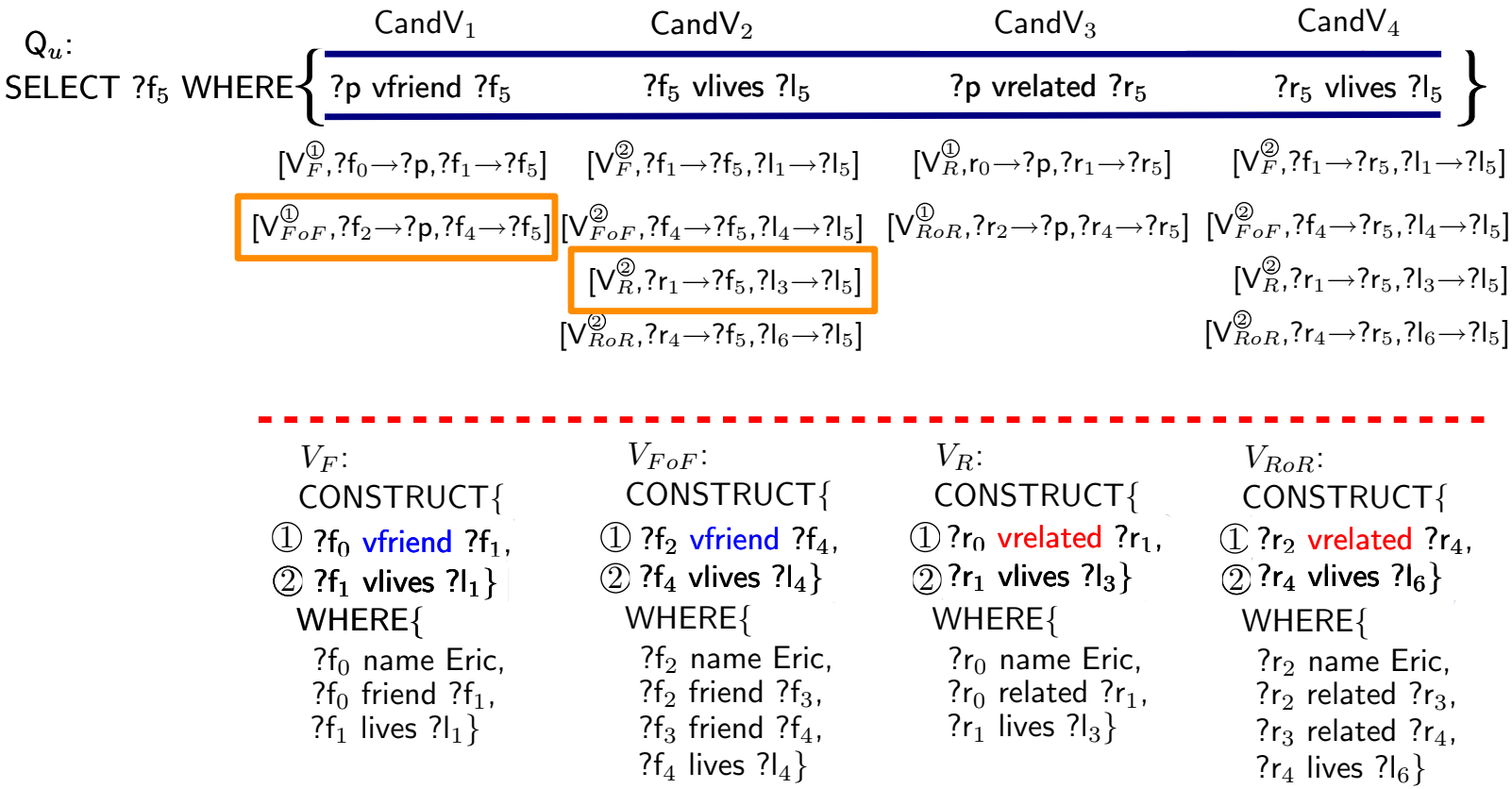
Theorem

Query q_{merge} resulting from replacing the two copies of the same view in rewritten query q with one is equivalent to q.

Outline

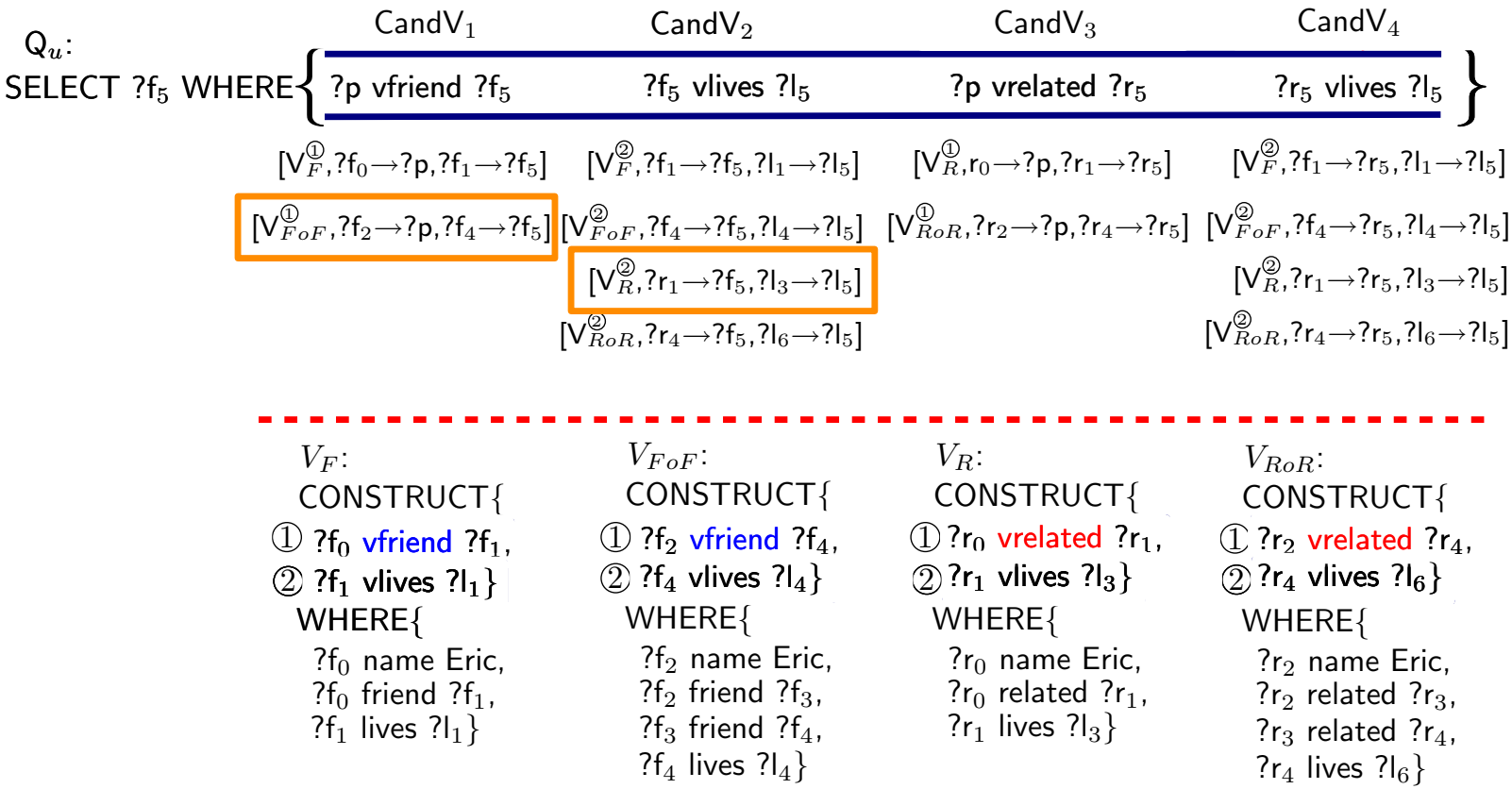
Optimization: pruning rewritings with empty results

- There are a subset of rewritings that join $V^{①}_{FoF}$ with $V^{②}_{R}$.

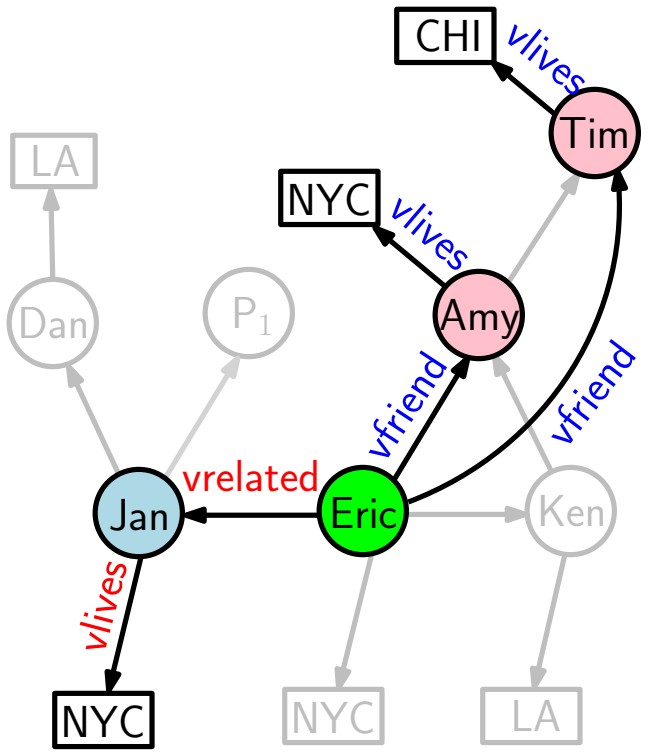


Optimization: pruning rewritings with empty results

- There are a subset of rewritings that join $V^{①}_{FoF}$ with $V^{②}_{R}$.
- Looking for Eric's friend-of-friend who is also his relative. Who are they?



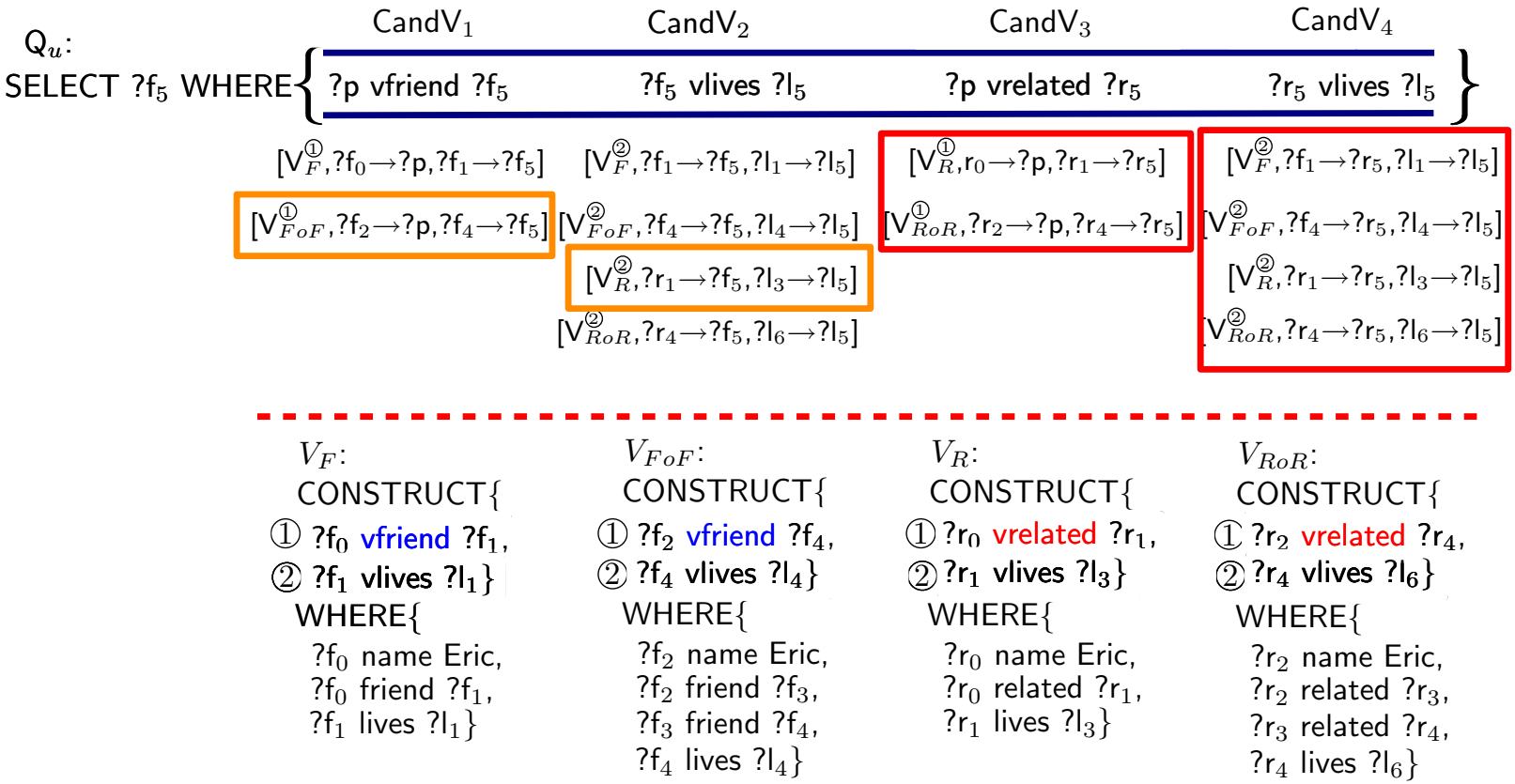
Optimization: pruning rewritings with empty results



- It turns out that Eric's FoF and R are disjoint.

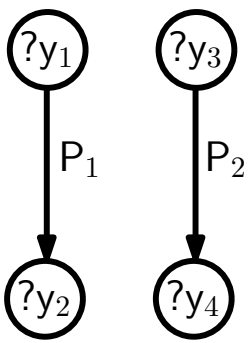
Optimization: pruning rewritings with empty results

- There are a subset of rewritings that join $V^{①}_{FoF}$ with $V^{②}_{R}$.
- Looking for Eric's friend-of-friend who is also his relative. Who are they?
- Prune all rewritings depending on the join between $V^{①}_{FoF}$ and $V^{②}_{R}$.



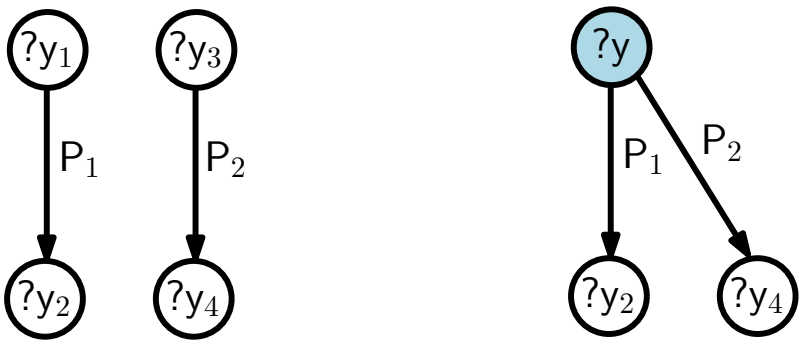
Optimization: pruning rewritings with empty results

- Consider a pair of triple patterns ($?y_1$, p_1, $?y_2$) and ($?y_3$, p_2, $?y_4$), suppose a join equals $?y_1$ and $?y_3$.



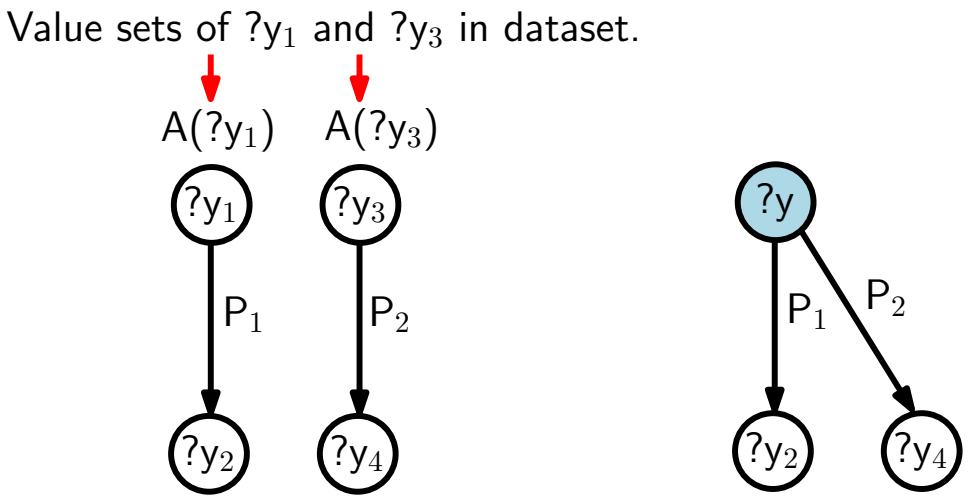
Optimization: pruning rewritings with empty results

- Consider a pair of triple patterns ($?y_1$, p_1, $?y_2$) and ($?y_3$, p_2, $?y_4$), suppose a join equals $?y_1$ and $?y_3$.



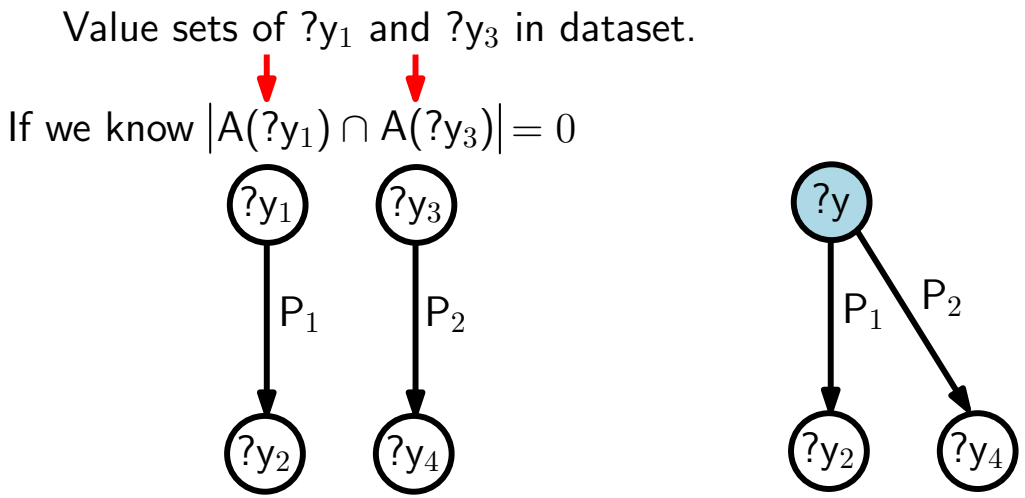
Optimization: pruning rewritings with empty results

- Consider a pair of triple patterns ($?y_1$, p_1, $?y_2$) and ($?y_3$, p_2, $?y_4$), suppose a join equals $?y_1$ and $?y_3$.



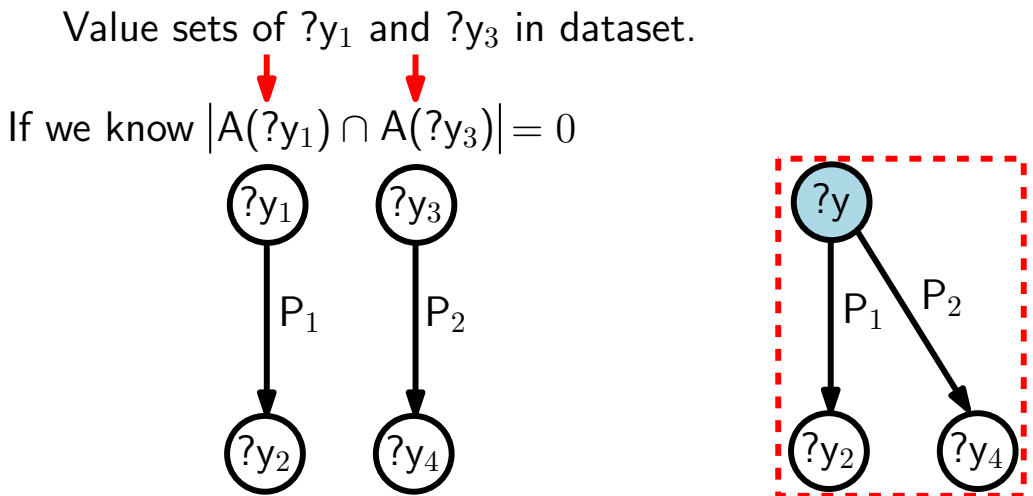
Optimization: pruning rewritings with empty results

- Consider a pair of triple patterns ($?y_1$, p_1, $?y_2$) and ($?y_3$, p_2, $?y_4$), suppose a join equals $?y_1$ and $?y_3$.



Optimization: pruning rewritings with empty results

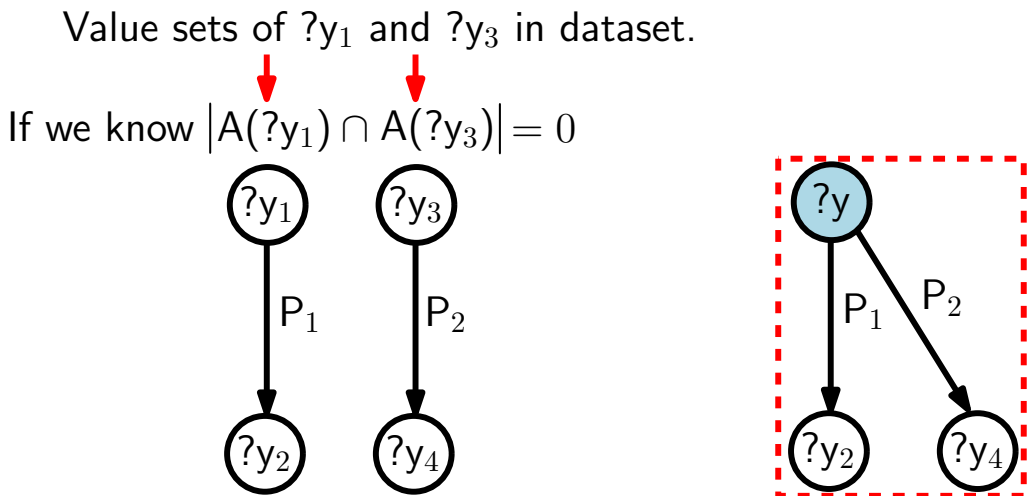
- Consider a pair of triple patterns ($?y_1$, p_1, $?y_2$) and ($?y_3$, p_2, $?y_4$), suppose a join equals $?y_1$ and $?y_3$.



- Any query with such join can be safely pruned.

Optimization: pruning rewritings with empty results

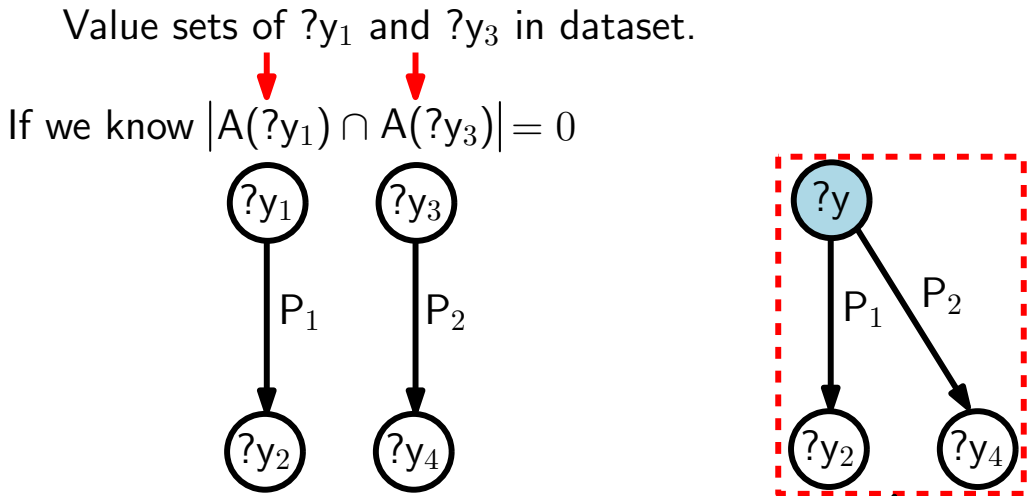
- Consider a pair of triple patterns ($?y_1$, p_1, $?y_2$) and ($?y_3$, p_2, $?y_4$), suppose a join equals $?y_1$ and $?y_3$.



- Any query with such join can be safely pruned.
- In theory, boolean intersection problem has been proven to requires linear space.

Optimization: pruning rewritings with empty results

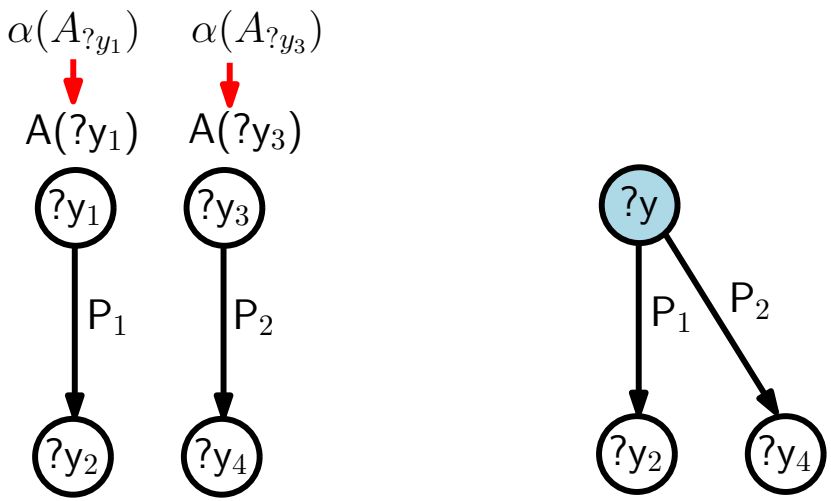
- Consider a pair of triple patterns ($?y_1$, p_1, $?y_2$) and ($?y_3$, p_2, $?y_4$), suppose a join equals $?y_1$ and $?y_3$.



- Any query with such join can be safely pruned.
- In theory, boolean intersection problem has been proven to requires linear space.
- In practice, we can settle to a constant success probability and thus design a space-efficient heuristic.

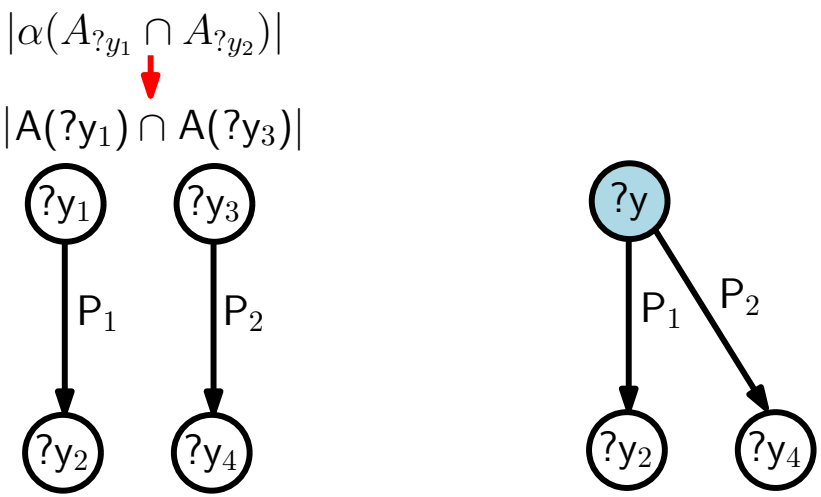
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



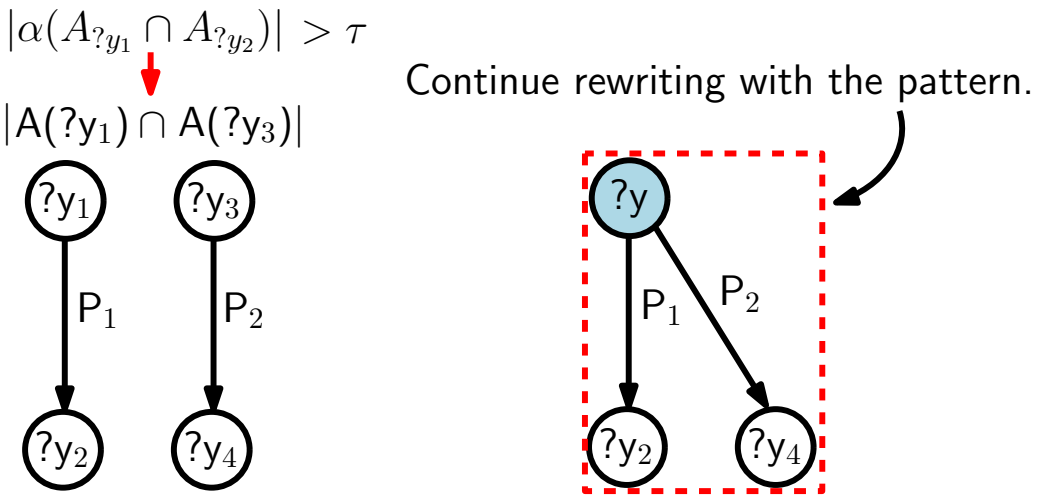
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



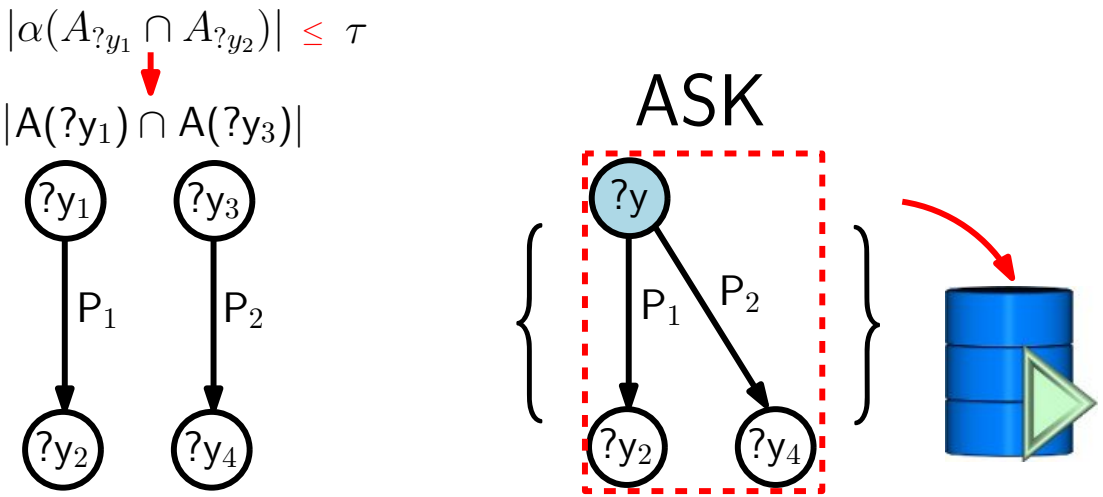
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



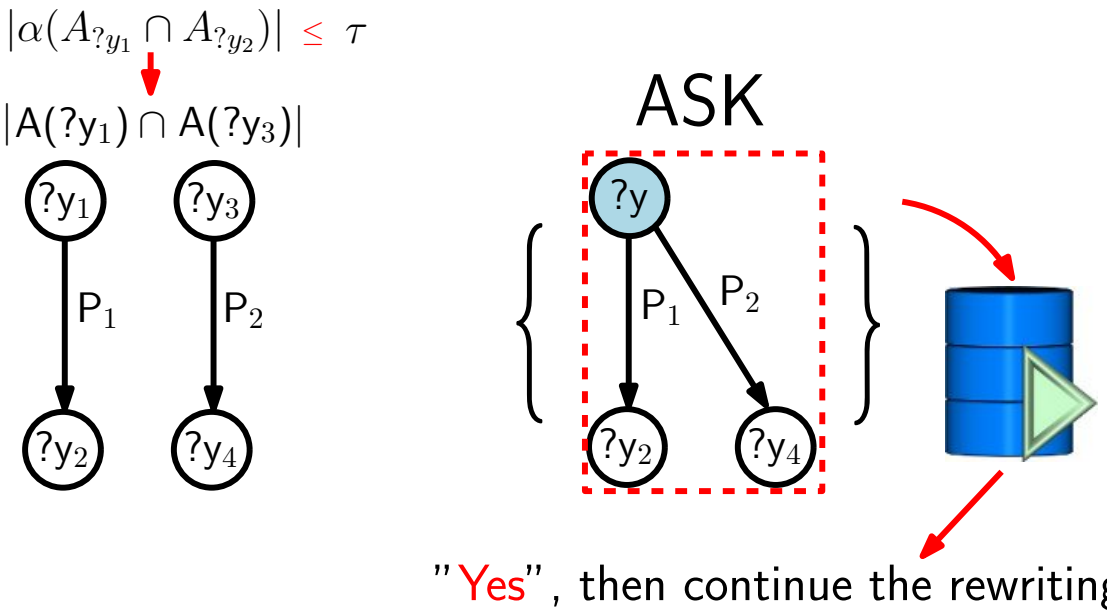
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



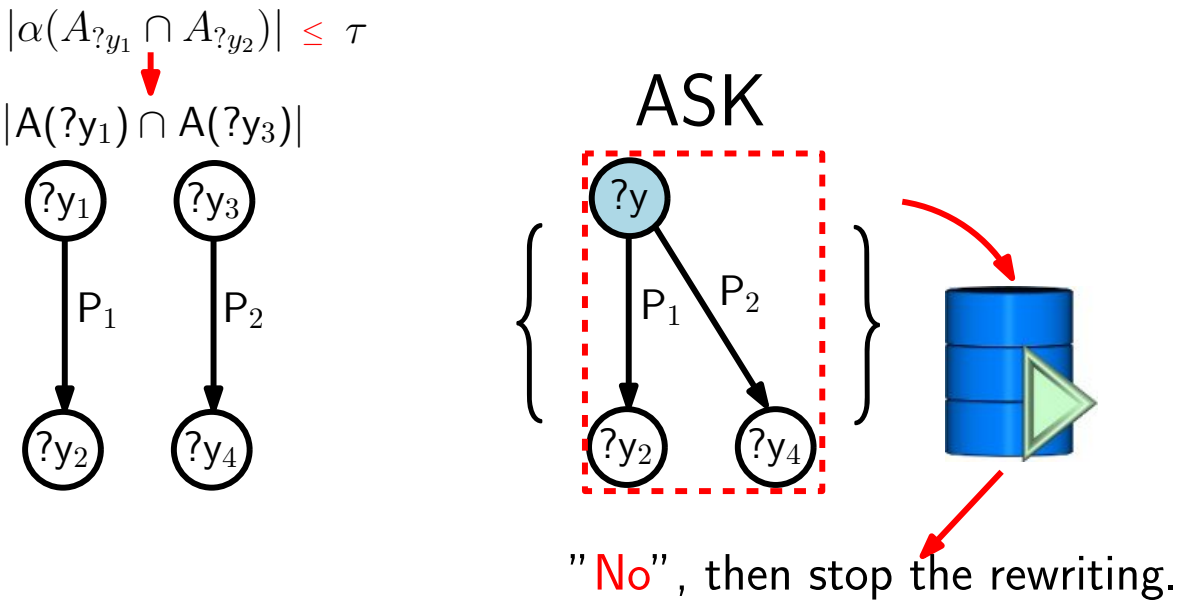
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



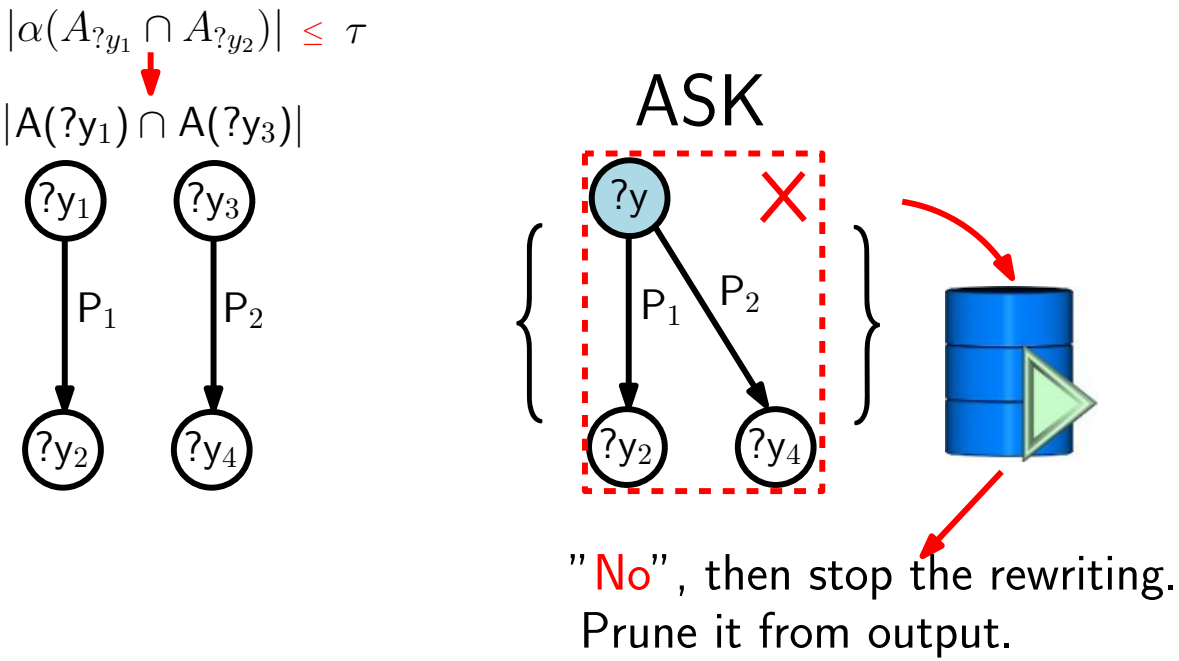
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



*In the paper, we presented heuristics to avoid issuing execessive ASK queries in rewriting.

Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.
- In practice, we use K Minimum Value (KMV) synopses to obtain unbiased estimations for the size of join.

Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.
- In practice, we use K Minimum Value (KMV) synopses to obtain unbiased estimations for the size of join.
- In the paper, we also discuss how to maintain the synopses, *i.e.*, dealing with updates.

Outline

Experiments

- Implementation: We implemented the rewriting algorithms and optimizations in C++.
- Views and Queries: We defined queries and views on LUBM benchmark (*e.g.* about courses, students, professors).
- Data: 10M triples from LUBM.
- Platform: 64-bit linux with 2GHz Intel Xeon CPU and 4GB Mem.
- Store: Evaluations were performed on MySQL (for relational exp) and 4store (a generic RDF store).

Experiments: Notation

- SPARQL query rewriting without optimizations as SQR.
- SPARQL query rewriting with optimizations as OSQR.
- Translation to SQL and rewriting in relational DB as SQL.
- Refer to our paper for the full set of experiments.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

Person 1 publish all his/her views.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Person 14 only published the view about name.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Q_u: SELECT * {
① ?x name ?n, ② ?x email ?e, ③ ?x DegreeFrom ?d,
}

- A query incrementally asks for more properties.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

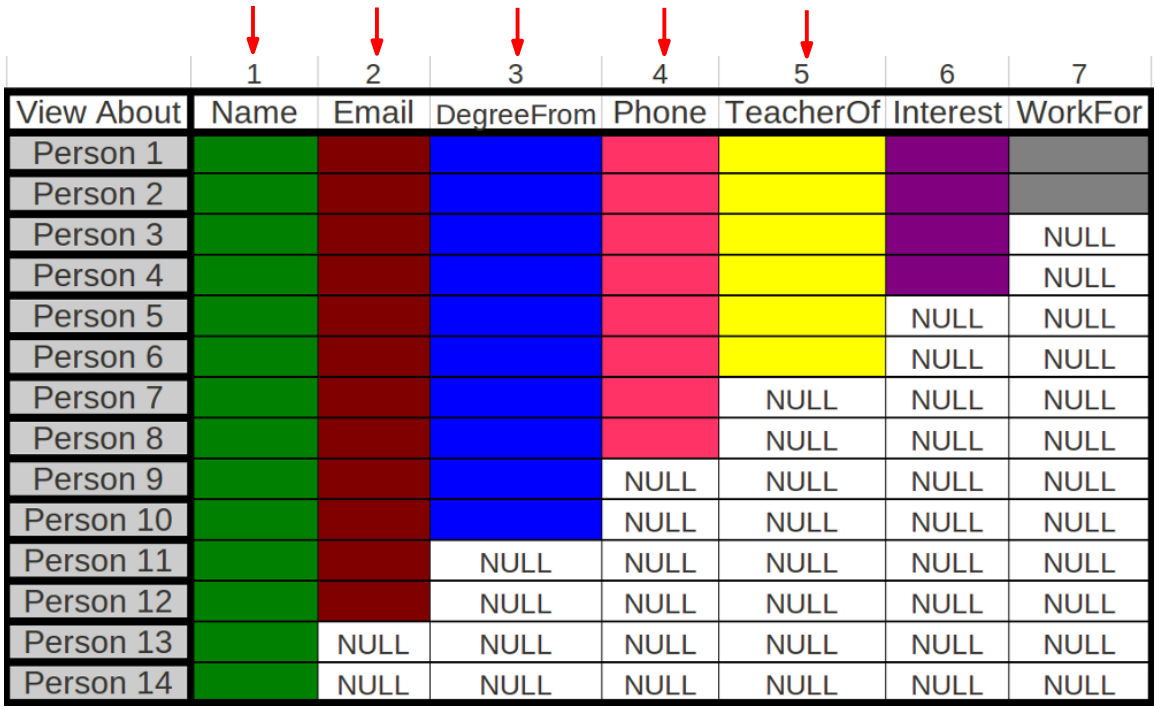
	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Q_u: SELECT * {
① ?x name ?n, ② ?x email ?e, ③ ?x DegreeFrom ?d, ④ ?x phone ?p,
}

- A query incrementally asks for more properties.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.



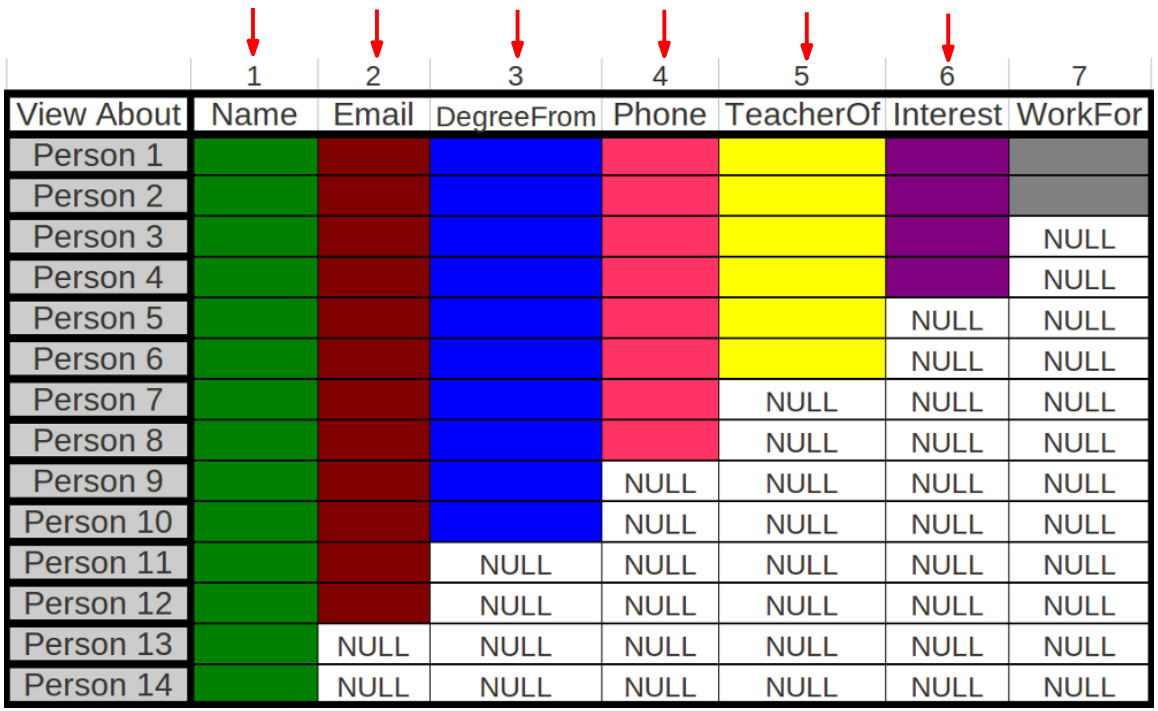
	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Q_u: SELECT * {
① ?x name ?n, ② ?x email ?e, ③ ?x DegreeFrom ?d, ④ ?x phone ?p,
⑤ ?x teacherOf ?t,
}

- A query incrementally asks for more properties.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.



	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

Q_u: SELECT * {
① ?x name ?n, ② ?x email ?e, ③ ?x DegreeFrom ?d, ④ ?x phone ?p,
⑤ ?x teacherOf ?t, ⑥ ?x interest ?i,
}

- A query incrementally asks for more properties.

Experiments: Setup

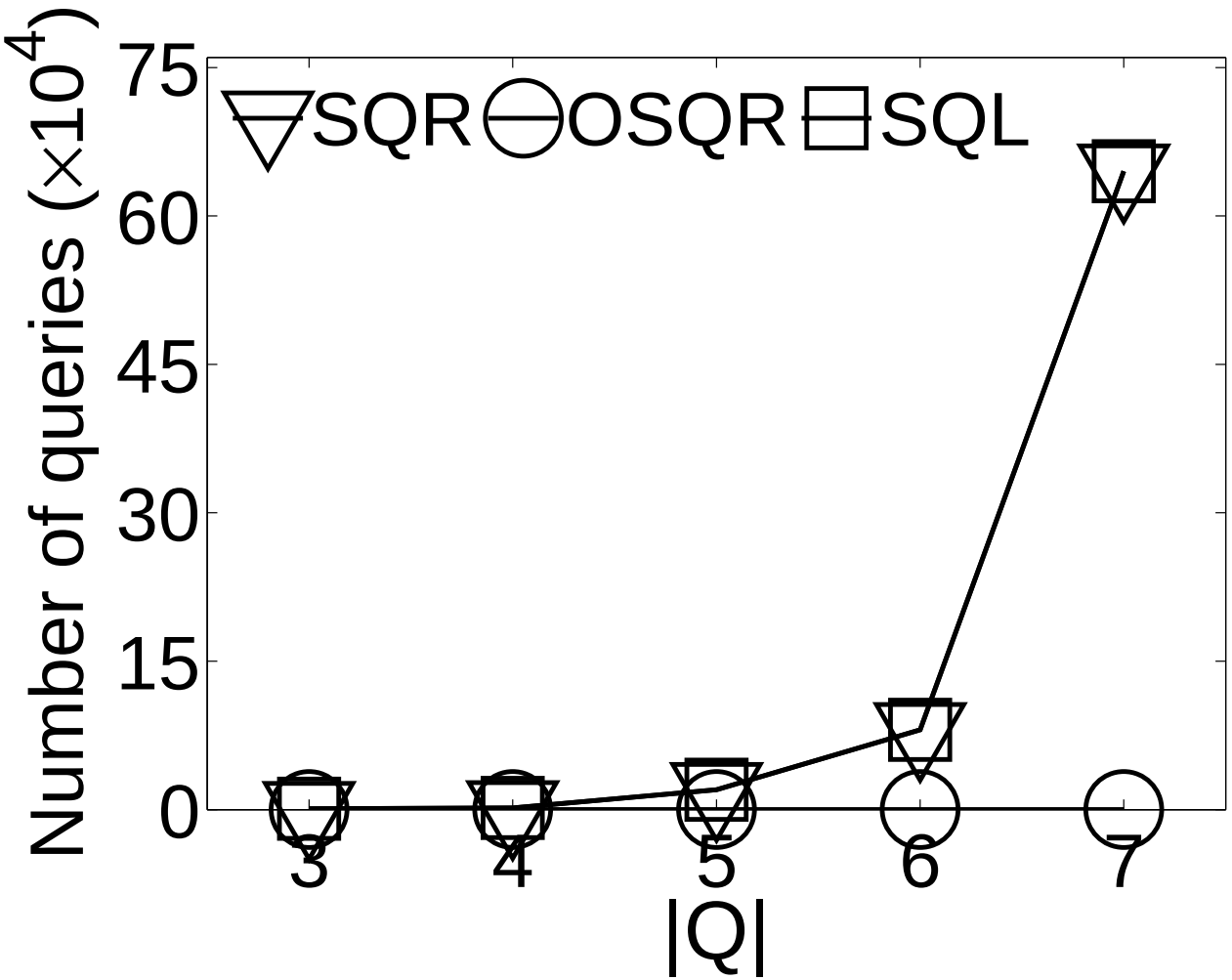
- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13		NULL	NULL	NULL	NULL	NULL	NULL
Person 14		NULL	NULL	NULL	NULL	NULL	NULL

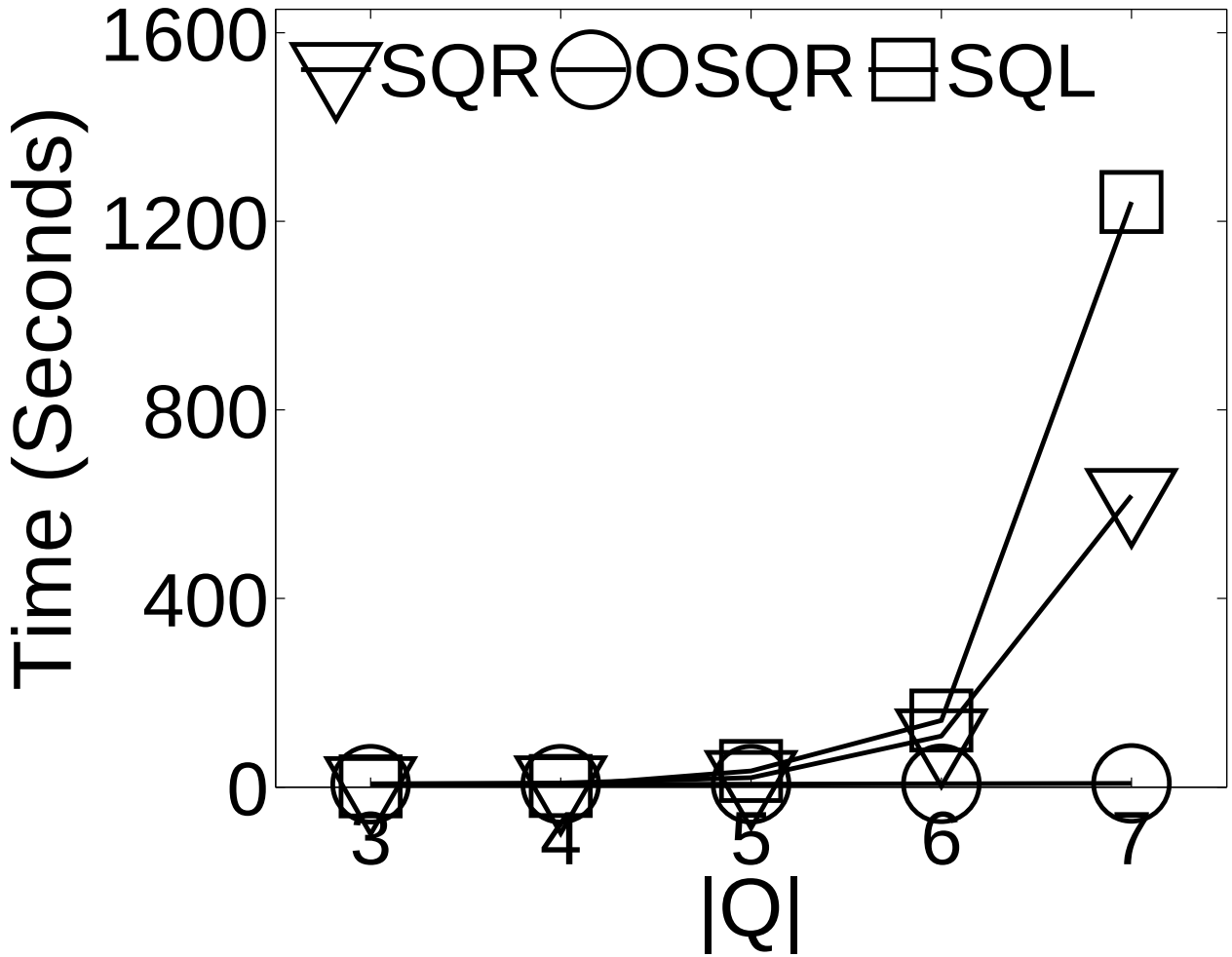
Q_u: SELECT * {
① ?x name ?n, ② ?x email ?e, ③ ?x DegreeFrom ?d, ④ ?x phone ?p,
⑤ ?x teacherOf ?t, ⑥ ?x interest ?i, ⑦ ?x worksFor ?w.
}

- A query incrementally asks for more properties.

Experiments: Results



Experiments: Results



Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	

Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL

Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL
Univ 6	NULL		NULL	NULL	NULL

Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL
Univ 6	NULL		NULL	NULL	NULL
Univ 7	NULL	NULL			NULL

Experiments: Setup

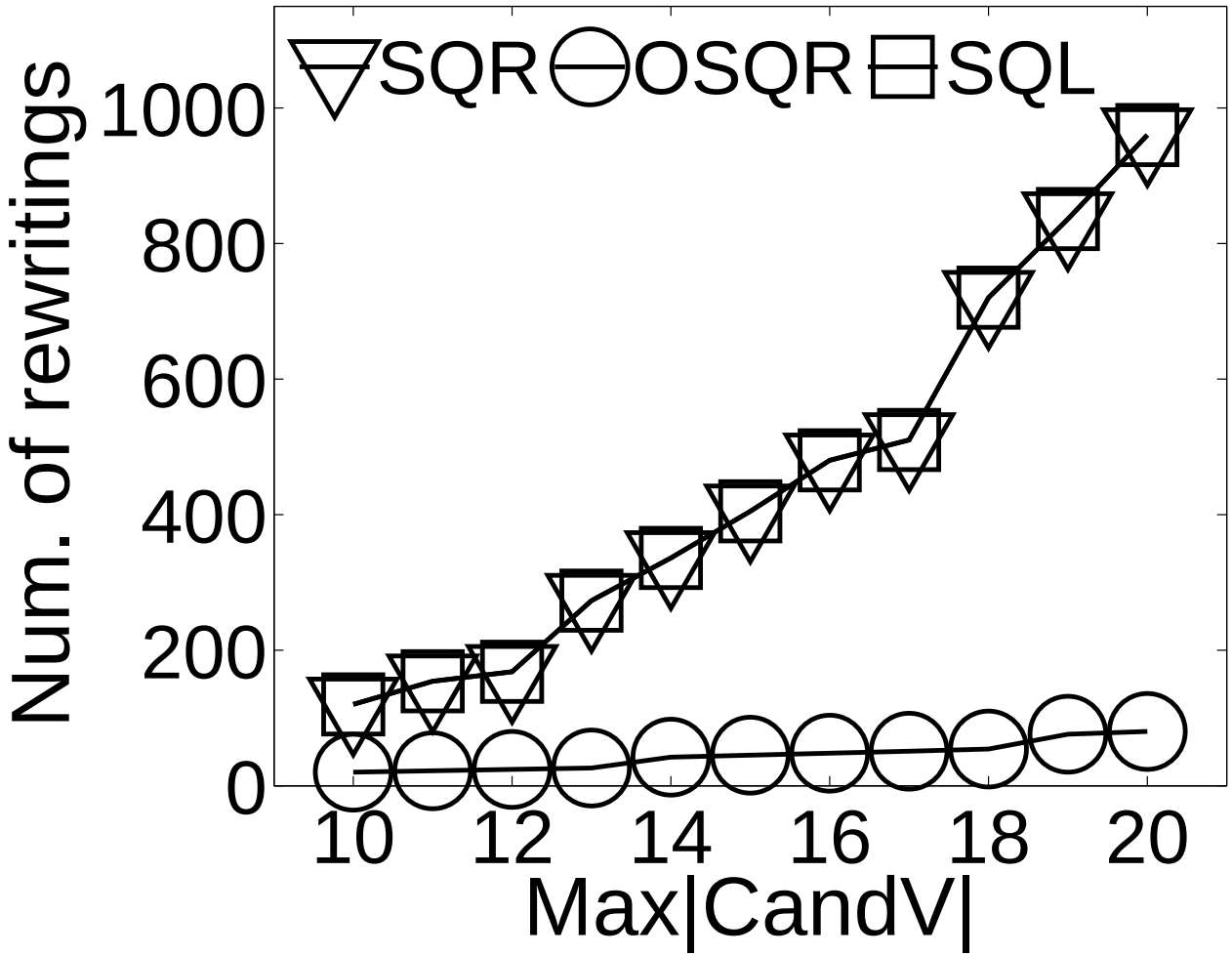
	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL
Univ 6	NULL		NULL	NULL	NULL
Univ 7	NULL	NULL			NULL
Univ 8	NULL	NULL	NULL	NULL	
Univ 9		NULL	NULL	NULL	NULL
Univ 10	NULL		NULL	NULL	NULL
Univ 11	NULL	NULL			NULL
Univ 12	NULL	NULL	NULL	NULL	
Univ 13		NULL	NULL	NULL	NULL
Univ 14	NULL		NULL	NULL	NULL
Univ 15	NULL	NULL			NULL
Univ 16	NULL	NULL	NULL	NULL	
Univ 17		NULL	NULL	NULL	NULL
Univ 18	NULL		NULL	NULL	NULL
Univ 19	NULL	NULL			NULL
Univ 20	NULL	NULL	NULL	NULL	

Experiments: Setup

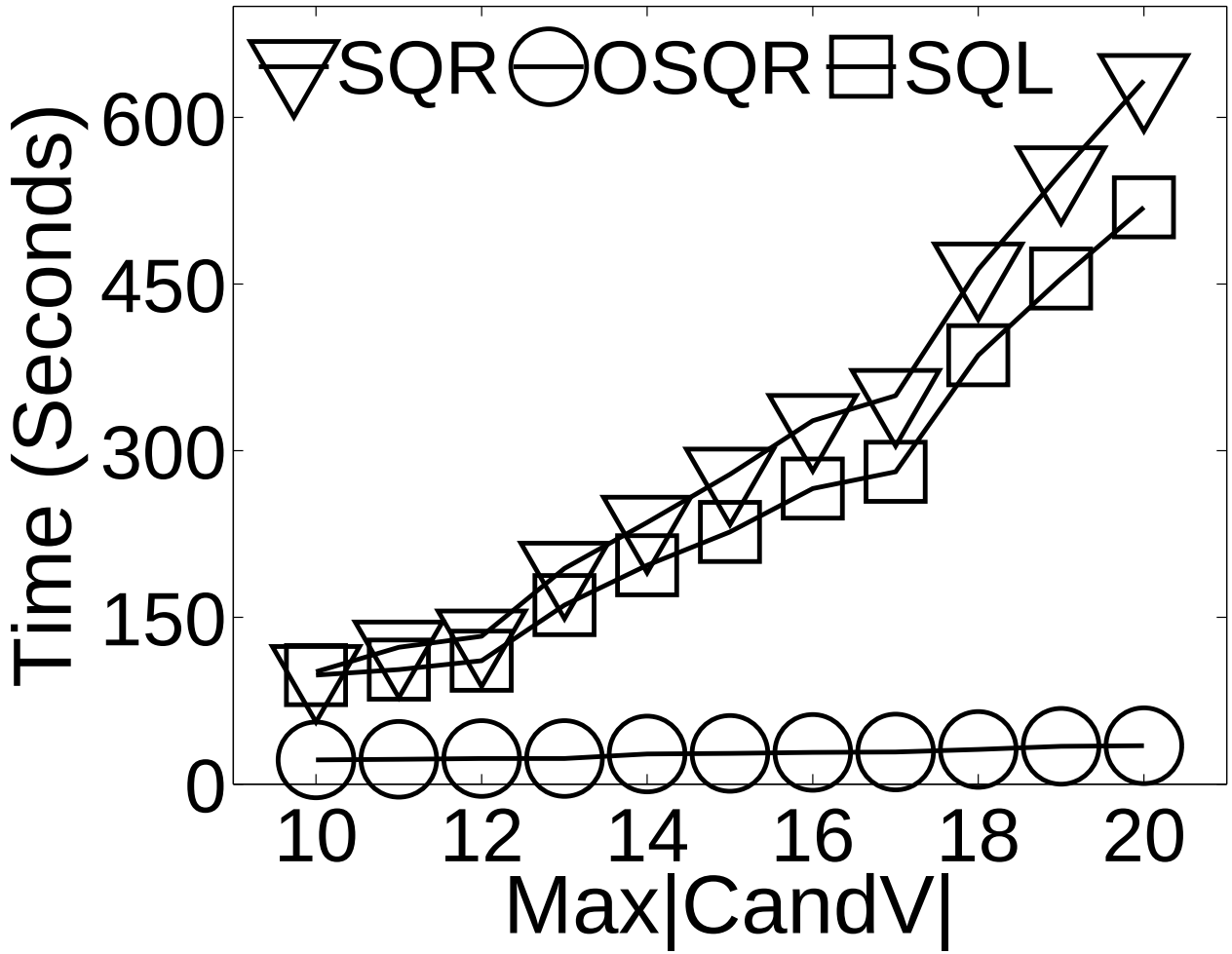
	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL
Univ 6	NULL		NULL	NULL	NULL
Univ 7	NULL	NULL			NULL
Univ 8	NULL	NULL	NULL	NULL	
Univ 9		NULL	NULL	NULL	NULL
Univ 10	NULL		NULL	NULL	NULL
Univ 11	NULL	NULL			NULL
Univ 12	NULL	NULL	NULL	NULL	
Univ 13		NULL	NULL	NULL	NULL
Univ 14	NULL		NULL	NULL	NULL
Univ 15	NULL	NULL			NULL
Univ 16	NULL	NULL	NULL	NULL	
Univ 17		NULL	NULL	NULL	NULL
Univ 18	NULL		NULL	NULL	NULL
Univ 19	NULL	NULL			NULL
Univ 20	NULL	NULL	NULL	NULL	

Q_u: SELECT * {
① ?x course ?n, ② ?x email ?e,
③ ?x degree ?d,
}

Experiments: Results



Experiments: Results



Outline

Conclusion

- We studied rewriting queries over views in the context of SPARQL and RDF.
- We proposed the first *sound* and *complete* rewriting algorithm.
- Novel optimizations to simplify individual rewritings and prune rewritings with empty results.
- Our solution is independent of RDF stores and hence portable.
- Extensive experiments demonstrate that our method is efficient and scalable.

The End

Thank You

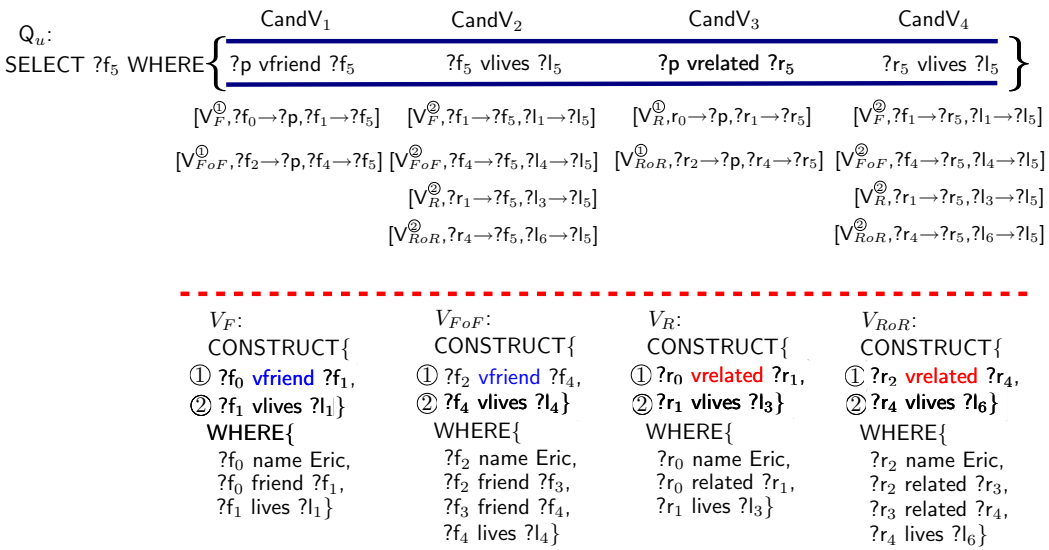
Q and A

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

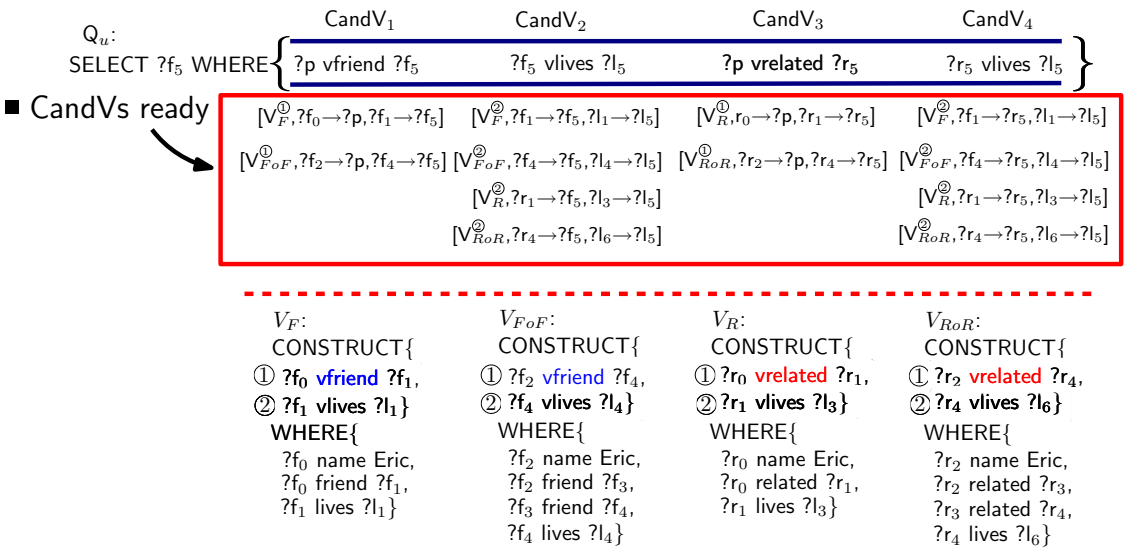
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



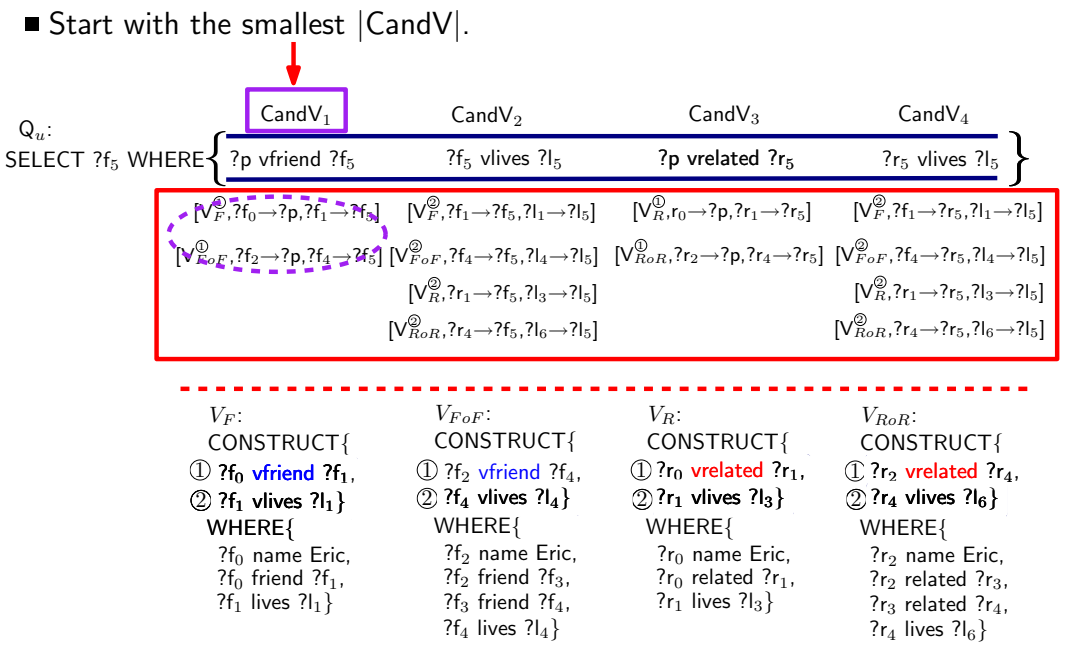
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



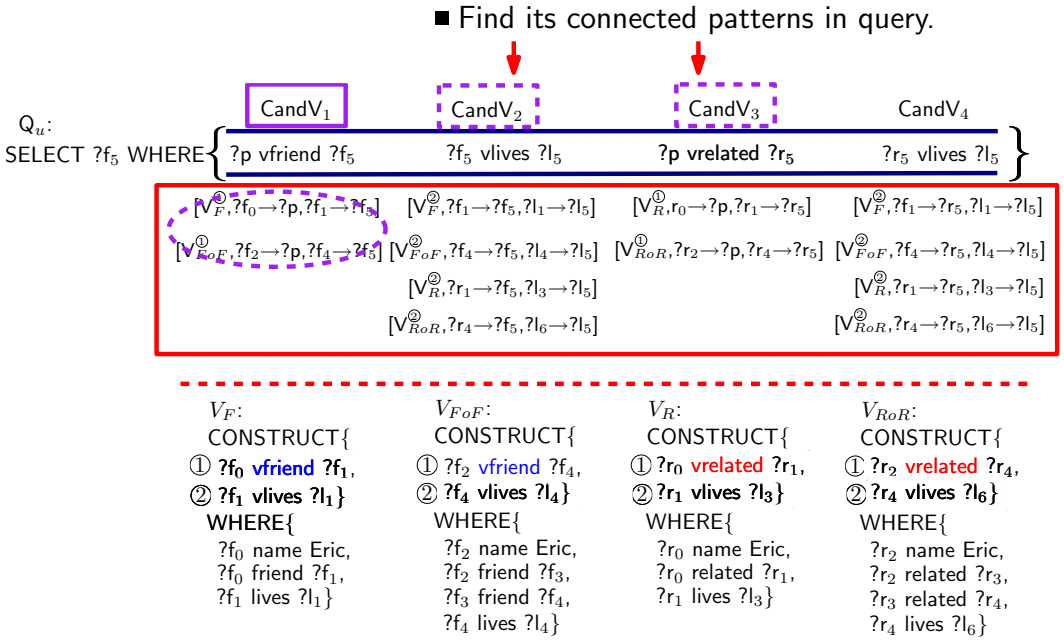
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



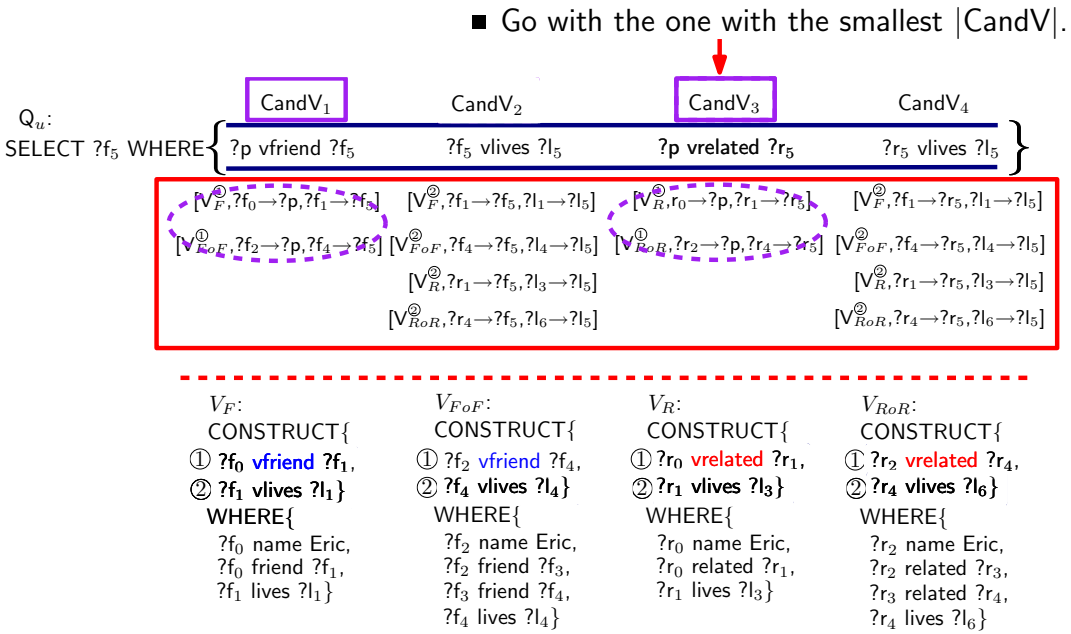
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



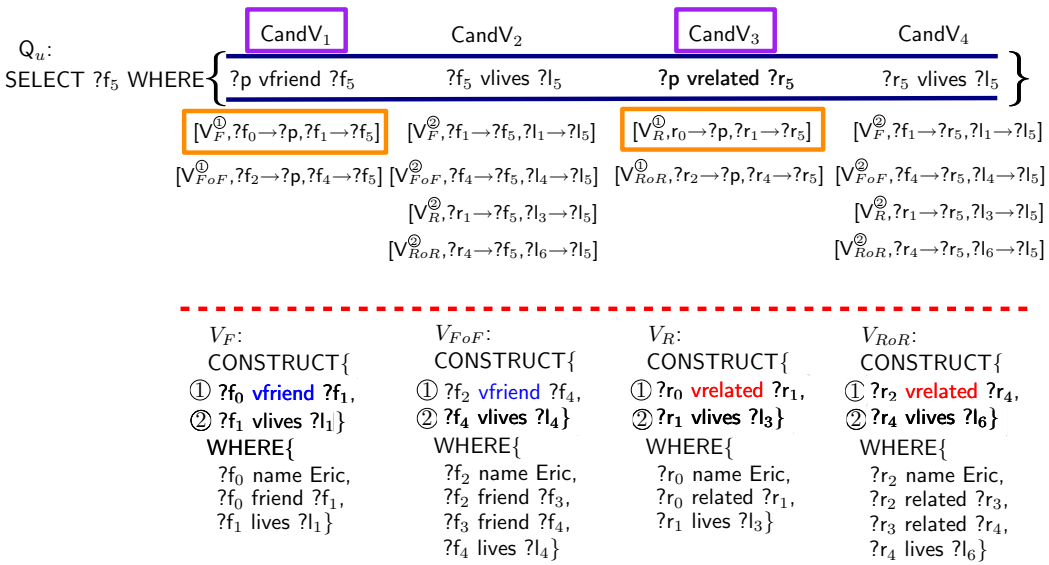
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



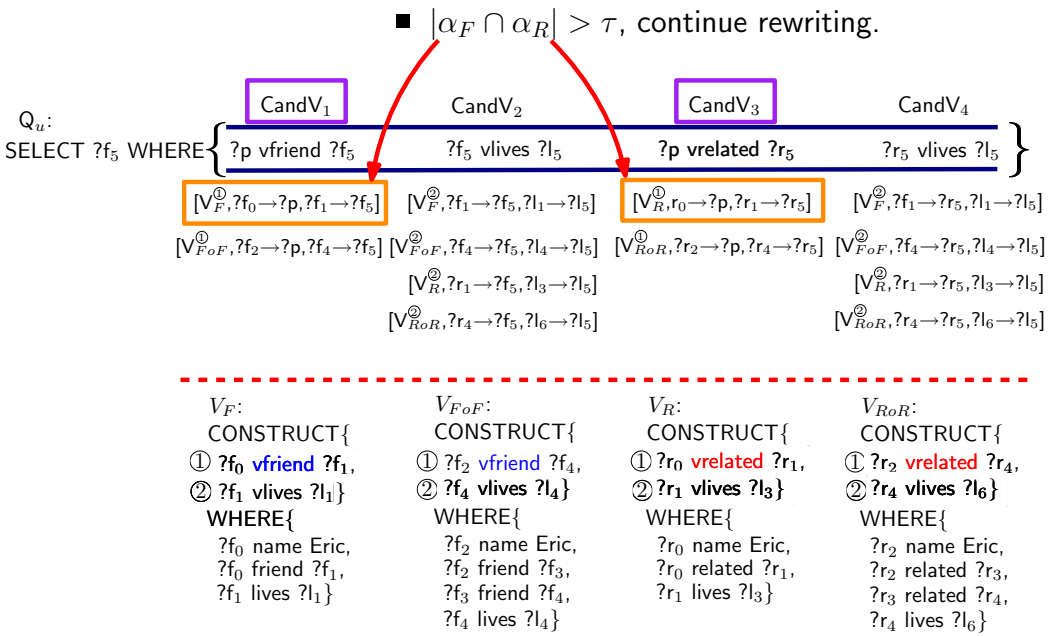
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



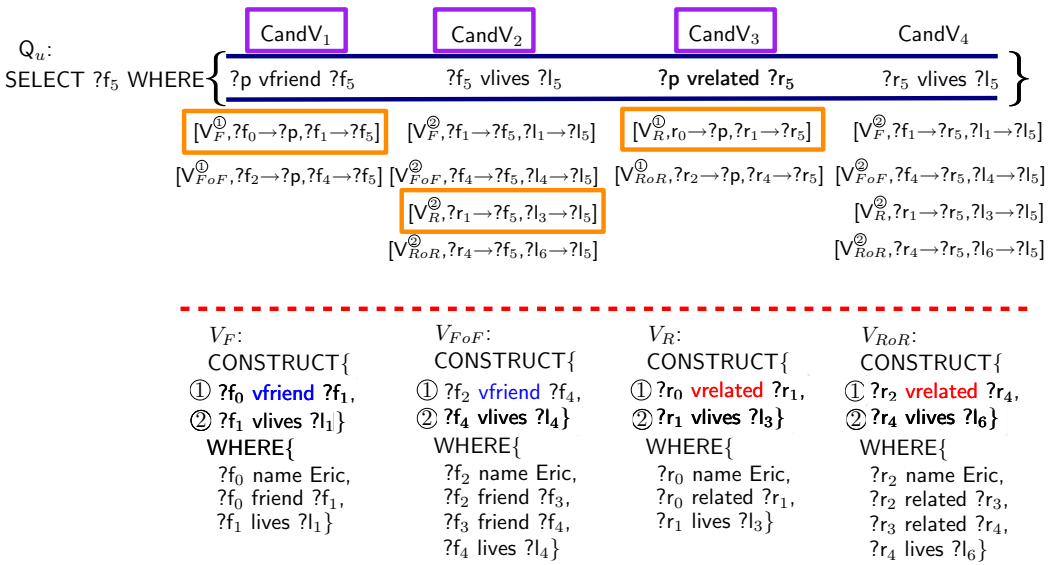
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



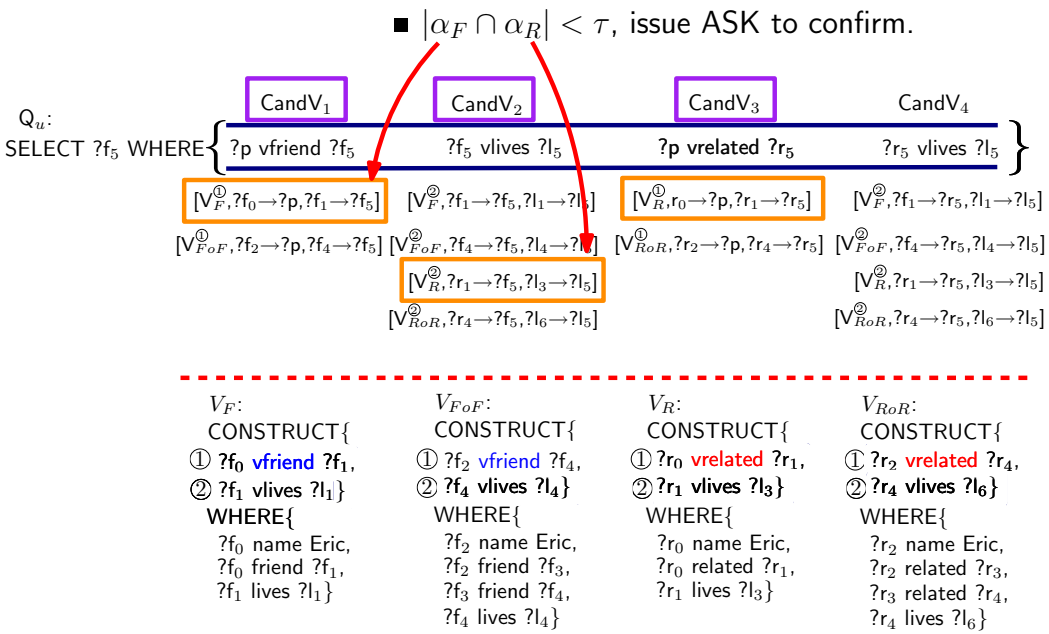
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.